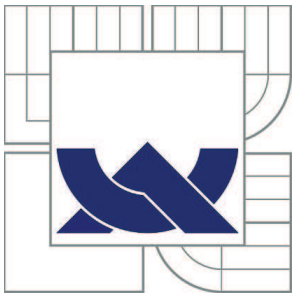
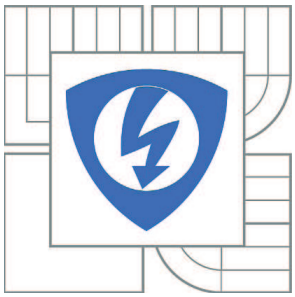


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

VIZUÁLNÍ DETEKCE ELEKTRONICKÝCH SOUČÁSTEK

VISUAL DETECTION OF ELECTRONIC DEVICES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

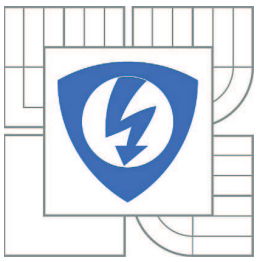
AUTOR PRÁCE — Bc. MIROSLAV JUHAS

AUTHOR

VEDOUCÍ PRÁCE — Ing. ILONA KALOVÁ, Ph.D.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Miroslav Juhas	***ID:***	78566
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Vizuální detekce elektronických součástek

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je seznámit se s problematikou řešeného projektu automatizované výroby "tuning fork with tip" (ladičky oscilátoru pro AFM mikroskopy). Cílem je návrh a implementace vhodných algoritmů detekujících v obrazech všechny potřebné části součástky při jednotlivých fázích zpracování. Dalším úkolem je vytvoření uživatelského prostřední umožňující snadnou obsluhu ale i nastavení všech potřebných parametrů a ladění algoritmů.

DOPORUČENÁ LITERATURA:

Hlaváč, V., Šonka, M.: Počítačové vidění. Praha: Grada, 1992.
Haußecker H., Geißler P.: Handbook of Computer Vision and Applications. San Diego: Academic press, 1999.
dále dle pokynů vedoucího

Termín zadání:	8.2.2010	***Termín odevzdání:***	24.5.2010

Vedoucí práce: Ing. Ilona Kalová, Ph.D.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Vizuální detekce elektronických součástek

Diplomová práce

Zaměření práce: Počítačové vidění
Student: Bc. Miroslav Juhas
Vedoucí práce Ing. Ilona Kalová, Ph.D.

Abstrakt:

Tato práce popisuje použití zpracování obrazu pro přesné měření vzdáleností v automatické výrobě hrotu pro AFM mikroskop. Hlavním cílem je měření vzdáleností mezi jednotlivými díly během výroby. Účelem je získání dat pro automatickou výrobní linku, která má nahradit nepřesnou a neopakovatelnou manuální výrobu.

Výrobní proces sestává ze tří technologických kroků. V prvních dvou krocích je přilepen wolframový drát k nosníku. V těchto krocích je, pro správné zarovnaní částí, nutno měřit vzdálenost ve všech třech osách. Ve třetím kroku je odleptán ostrý hrot v roztoku KOH. Musí být dodržena správná vzdálenost mezi hladinou roztoku a nosníkem.

K získání obrazu je použita kamera s vysokým rozlišením a makro objektivem. Obraz je poté kalibrován, aby bylo odstraněno zkreslení a vliv polohy scény vzhledem k umístění kamery. Je také zjištěn délkový převodní koeficient. Rozpoznání objektů a měření vzdálenosti využívá standardní metody počítačového vidění: adaptivní prahování, momenty, statistické vlastnosti obrazu, Cannyho hranový detektor, Houghovu transformaci,…

Navržené algoritmy byly implementovány v jazyce C++ s použitím Intel OpenCV knihovny. Finální dosažené rozlišení při měření je 10 μm na pixel. Výstup algoritmů byl použit k sestavení několika testovacích hrotů.

Klíčová slova: kalibrace kamery, korekce zkreslení kamery, měření vzdálenosti v obrazu, rozeznání průmyslových objektů, objektový softwarový návrh, AFM hrot

Visual detection of electronic devices

Master's thesis

Specialization of study: Computer vision
Student: Bc. Miroslav Juhas
Supervisor: Ing. Ilona Kalová, Ph.D.

Abstract:

This thesis describes application of image processing for precise distance measurement in self acting production of a tip for AFM microscopes. The main goal is to measure distances between assembly parts during fabrication process. The purpose is to acquire a data for self acting assembly line which have to substitute inaccurate and nonrecurring manual assembly process.

The assembly process consists of three technological steps. In first two steps the tungsten wire is glued to the cantilever. Distance measurement is necessary in all axes for proper alignment of parts. In third step the sharp tip is etched by KOH solution. The right distance between liquid level and the cantilever must be kept.

A camera with high resolution and macro objective is used to acquire an image. Camera image is then calibrated to suppress distortions and scene position with respect to camera position. Length conversion coefficient is also computed. Object recognition and distance measurement is based on standard computer vision methods, mainly: adaptive thresholding, moments, image statistics, canny edge detector, Hough transform…

Proposed algorithms have been implemented in C++ using Intel OpenCV library. The final achieved distance resolution is about 10µm per pixel. Algorithm output was successfully used to assembly few test tips.

Keywords: camera calibration, camera distortion correction, image distance measurement, industrial object recognition, object oriented software design, AFM tip

Bibliografická citace:

JUHAS, M. *Vizuální detekce elektronických součástek.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 68 s. Vedoucí diplomové práce Ing. Ilona Kalová, Ph.D.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Vizuální detekce elektronických součástek jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **24. května 2010** ……
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Iloně Kalové, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **25. května 2010** ……
podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1. ÚVOD

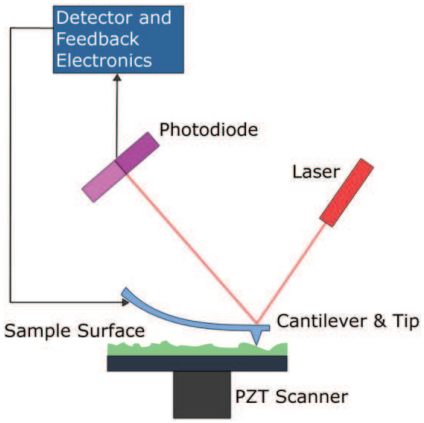
Úkolem této práce, zaměřené na problematiku počítačového vidění, je detekce částí elektronické součástky při montáži na výrobní lince. Sestavovanou součástkou je hrot pro mikroskop atomárních sil (AFM – Atomic force microscope).

AFM nebo také SFM (scanning force microscope) je skenovací mikroskop s velmi velkou rozlišovací schopností s rozlišením ve zlomcích nanometrů, více než 1000krát lepším než je difrakční limit optických mikroskopů. Předchůdce AFM, skenovací tunelový mikroskop, vyvinul Gred Binning a Heinrich Rohrer v osmdesátých letech. Vývoj jim vynesl Nobelovu cenu za fyziku v roce 1986. Binnig, Quate a Gerber vynalezli první AFM v roce 1986. AFM lze použít nejen k zobrazování, ale také k tvorbě struktur či zpracování povrchů v nanometrové oblasti.([9])

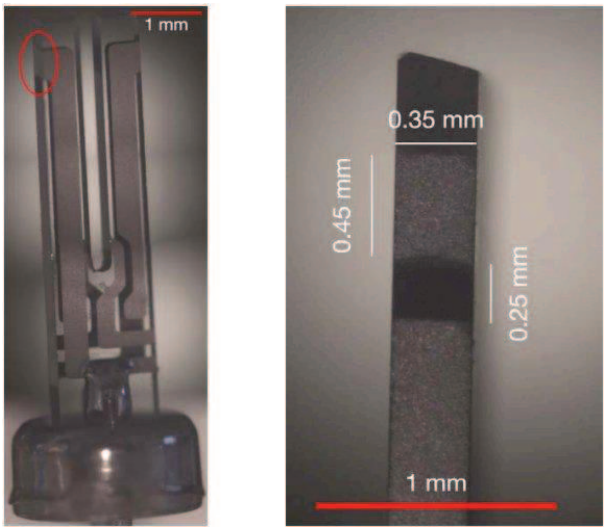
Základem AFM je ostrý hrot, upevněný na ohebném nebo pevném nosníku. Při snímání se používá kontaktní nebo nekontaktní způsob. Při kontaktním snímání se hrot pohybuje po měřeném povrchu, což však může vést k jeho poškození nebo k poškození hrotu. Při bezkontaktním režimu se hrot pohybuje nad měřeným povrchem a je rozkmitáván (opačný piezoelektrický jev). Kmitání je tlumeno působením Van der Waalsovy síly působící mezi hrotem a povrchem. Velikost tlumení je dána vzdáleností mezi povrchem a hrotem a je tedy možno měřit nerovnosti povrchu. Snímání pohybu hrotu se často děje laserovým paprskem, který se odráží od nosníku na fotodetektor.(Obr. 1) Ve špičkových aplikacích se používá paprsek rentgenového záření. Tento popsaný princip je pouze jednou z možností provedení, ve skutečnosti existuje velké množství různých principů, typů hrotů i snímání.

Small Infinity je začínající spin-off ESRF (European Synchrotron Radiation Facility, Francie) specializující se na vývoj AFM mikroskopů. V současné době zde montáž specifických hrotů pro AFM mikroskop vlastní výroby provádí ručně v jednotkách kusů. Vzhledem k velikosti součástky (Obr. 2) a tloušťce drátku (průměr 0.08 mm), který se stane po odleptání hrotem, je tento postup náročný (nutnost montáže pod mikroskopem) a zdlouhavý (přibližně 6 minut).

Hrot je prakticky spotřebním zbožím, ačkoliv v jednotkách kusů denně. Ruční výroba není dostatečně přesná a opakovatelná, což vede k nutnosti kalibrace mikroskopu na každý nový hrot samostatně. Účelem této práce je proces automatizovat, respektive poskytnout data pro výrobní linku provádějící montáž hrotu podobným způsobem, jako by byla prováděna manuálně.



Obr. 1: Princip AFM [9]



Obr. 2: Nosník pro motáž hrotu a jeho rozměry (autor O. Dhez, Small Infinity)

Výrobní proces sestává z několika technologických kroků. Prvním krokem je nanesení vodivého lepidla na plošku o rozměrech 0.35x0.45 mm na boku nosníku (Obr. 2), poté se pomocí tohoto lepidla přilepí k nosníku wolframový drátek. Další krok je leptání drátku v roztoku KOH tak, aby vznikl ostrý hrot. Leptání probíhá

v leptacím roztoku průchodem elektrického proudu. Hrot musí být co nejkratší a vrstva lepidla co nejmenší, aby jejich hmotnost nosník při oscilacích co nejméně tlumila. Při leptání však nesmí dojít ke kontaktu lepidla a roztoku. Stejně tak nesmí dojít ke spojení vodivých cest na nosníku při nanášení lepidla.

Z důvodu velmi malých rozměrů je při automatizaci výrobního procesu použití standardních snímačů polohy problematické. Oko člověka by v tomto případě mělo být nahrazeno kamerou s příslušnou optikou, dostatečným rozlišením a doplněnou algoritmy schopnými rozlišit jednotlivé objekty pro montáž a jejich vzájemnou polohu.

Hlavním výstupem a cílem práce je vytvoření softwaru zahrnujícího uživatelské rozhraní, logiku a řízení výroby a algoritmy zpracování obrazu v jednotlivých fázích výroby.

Stručný obsah následujících kapitol:

- Kapitola 2 obsahuje popis technických prostředků použitých při řešení projektu a to jak software tak hardware.
- Kapitola 3 se zabývá architekturou vytvořeného softwaru po programové stránce. Jsou uvedeny nároky kladené na vytvářený software a popis postupu vedoucího k jejich splnění. Jsou zde také prezentovány jednotlivé funkční části a architektura projektu.
- Kapitola 4 je rozborem řešení úkolů zpracování obrazu v projektu. Jsou zde uvedeny postupy předzpracování, segmentace, popisu objektů a principy kalibrace kamery, korekce zkreslení a měření vzdáleností v obrazu. Dále jsou uvedeny postupy zpracování obrazu při jednotlivých technologických krocích a jejich výsledná spolehlivost.
- Kapitola 5 shrnuje obsah práce a dosažené výsledky.

2. TECHNICKÉ PROSTŘEDKY

K řešení a zpracování uvedené úlohy je potřeba poměrně široká škála hardwarových a softwarových prostředků. Obor počítačového vidění je oborem, kde se nejedná pouze o algoritmy zpracování obrazu. Součástí počítačového vidění jsou také znalosti z jiných oborů, jako je matematika, optika, hardware pro získání a zpracování obrazu, strojové učení, výpočetní a softwarové prostředky, systémy pro přenos a uchovávání informací, zpracování signálů a další.

Tato kapitola je seznámením s hardwarem a softwarem, který byl použit při vypracování projektu. Programové řešení všech částí řízení a zpracování obrazu je implementováno v jazyce C++ pod OS Windows.

2.1 SNÍMACÍ ZAŘÍZENÍ

Jako zařízení pro pořízení obrazu byla použita barevná 2Mpx CMOS kamera společnosti FCLab připojená přes rozhraní USB 2.0. Některé hlavní vlastnosti zařízení:

- Rychlost – 12fps při 1600×1200, 16fps při 1280×1024, 20fps při 800×600.
- ROI (region of interest) – možnost výběru pozice a velikosti oblasti, která nás zajímá ve snímané scéně.
- Řízení expozice v rozsahu 1-500 ms.
- API software poskytovaný výrobcem.
- závit na objektiv, externí výstupy pro ovládání světel/blesků a další.

Hlavní výhodou je rychlost, řízení expozice a funkce ROI. ROI umožňuje přečíst ze snímače kamery jen tu oblast obrazových dat, která je zajímavá. Vzhledem k tomu, že rychlost získání snímku je závislá na době expozice a množství dat, které je nutno přenést, je možnost získání jen části dat, bez nutnosti zpracování celého obrazu, významnou výhodou šetřící čas i výpočetní prostředky.

Nevýhodou, kterou tato kamera přináší, je skutečnost, že přímo nespolupracuje s funkcemi pro získání obrazu knihovny OpenCV. Protože kamera

poskytuje nadstandardní funkce, které jsou součástí aplikačního rozhraní kamery, byla nutná implementace třídy zprostředkovávající rozhraní mezi knihovnou OpenCV a aplikačním rozhraním FCApi výrobce kamery. Tato třída poskytuje metody pro získání obrazu a pro řízení kamery jednoduše a transparentně, bez nutnosti zaobírat se „režií“ kolem získávání obrazu.

2.2 OPTIKA

Pro snímání obrazu v takto malých rozměrech je nutné doplnit kameru odpovídající optikou. V tomto případě objektivem Computar MLH10X. Objektiv disponuje zvětšením až 10:1. Clona, makro a ostření je manuální. Při celém výrobním procesu se předpokládá pevné neměnné zaostření i clona. Náhrada automatickým objektivem se nepředpokládá.

2.3 POLOHOVACÍ ZAŘÍZENÍ

K pohybu magnetického úchytu s nosníkem po výrobní lince je použit kontrolér krokových motorů s připojením přes sběrnici USB. V kombinaci s třemi velice přesnými krokovými motory s rozlišením až 1/8 kroku a krokem délky 2,5µm a 1,25 µm je zařízení schopné operovat s mikrometrickou přesností ve všech třech osách. Další stupeň volnosti je získán přidáním servomotoru pro natáčení magnetického úchytu nosníku kolem jeho osy.

2.4 OSTATNÍ HARDWARE

Kromě krokových motorů, pro hlavní tři osy pohybu ramene s nosníkem, je nutné další polohování a ovládání pomocí servomechanizmů, stejně jako zpracování signálů z různých čidel montážní linky. Z tohoto důvodu je nutný dodatečný hardware.

Pro řízení servomechanizmů a další vstupy a výstupy je použita deska založená na GNU/GPL projektu Arduino, jejímž základem bývá některý z mikrokontrolérů Atmel (v tomto případě konkrétně ATmega328). Zařízení komunikují prostřednictvím sériové linky (RS232), což je poměrně výhodné z hlediska jednoduchého programování jeho ovládání. Použitý hardware není čistě

průmyslový, ale vzhledem k prostředí a určení montážní linky je postačující. Výhodou je nízká cena, jednoduchost a dobrá dostupnost.

2.5 INTEL OPEN COMPUTER VISION LIBRARY – OPENCV

Při zpracování obrazu se pracuje s velkým množstvím dat, stejně tak se používá množství algoritmů pro zpracování těchto obrazových dat. Algoritmy pro počítačové vidění jsou většinou dobře popsány v odborné literatuře nebo na internetu. Implementace v některém z programovacích jazyků nebývá složitá. Co je však problémem, je rychlost. Optimalizace i obyčejné, běžně používané konvoluce, může být poměrně složitá záležitost, jejíž zpracování vydá na samostatný článek.

Protože je při zpracování obrazu použito velké množství algoritmů a často se experimentuje s různými postupy, je optimalizace rychlosti zpracování poměrně složitým problémem, obzvláště pokud jsou kladeny protichůdné nároky na rychlost a zároveň spolehlivost. Pro tento problém existuje řešení v podobě programových knihoven pro práci s obrazem, které mohou při vývoji značně ulehčit práci s psaním algoritmů i optimalizací rychlosti. Jednou z těchto knihoven je Intel OpenCV Library (dále jen OpenCV).

Intel OpenCV je open source knihovna pro počítačové vidění dostupná z http://SourceForge.net/projects/opencvlibrary. Knihovna je napsána v C a C++ a funguje pod operačními systémy Linux, Windows a Mac OS.([1]) Knihovna OpenCV je zaměřená na real-time zpracování obrazu a je použitelná jak pro výzkum, tak pro průmyslové aplikace. Pro další optimalizaci je možno použít knihovny IPP (Integrated Performance Primitives) společnosti Intel, zde je však již nutno zaplatit. OpenCV pokrývá širokou paletu problémů počítačového vidění a algoritmů. Obsahuje více než 500 funkcí. Kromě počítačového vidění také obsahuje plnohodnotnou knihovnu pro strojové učení MLL (Machine Learning Library).

2.6 MS VISUAL C++ 2008 EXPRESS EDITION

Pro vývoj aplikací a programování algoritmů v jazyce C/C++ je nutné odpovídající vývojové prostředí a kompilátor. Komerční řešení, jako je Borland C++ Builder nebo MS Visual Studio, jsou velice kvalitní nástroje s podporou ze strany

výrobce, jejich nevýhodou je většinou vysoká cena. Naproti tomu volné vývojové nástroje, jako je například IDE DEV C++ (grafické rozhraní a kompilátor GCC), jsou zdarma, uživatelská základna bývá široká, ale chybí zde širší podpora a komfort placených aplikací.

Jako vývojové prostředí bylo nakonec zvoleno MS Visual C++ 2008 Express Edition. Toto řešení je zlatou střední cestou, protože kombinuje pohodlí placených vývojových prostředí s nulovou cenou volných vývojových nástrojů. Výhodou je také možnost použití jak pro komerční tak nekomerční účely, obojí zdarma (Microsoft uvolnil zdarma Express edice svých vývojových nástrojů v roce 2006).

3. ARCHITEKTURA SOFTWARU

3.1 NÁROKY NA NÁVRH ŘÍDÍCÍHO SOFTWARU

Během provozu se může změnit hardware nebo technologie. Proto by měl být software navržen modulárně, nejlépe objektově. Stejně tak se mohou změnit parametry snímané scény. I když se může jednat pouze o malou změnu, pokud bude software navržen „napevno“ bez možnosti jednoduché úpravy parametrů zpracování uživatelem, pak znamená tato malá změna nemalé problémy při změnách kódu aplikace.

Aby byla nutnost složitých zásahů do kódu při změně podmínek minimalizována, je nutné při vývoji řídící aplikace a zpracování obrazu dodržet určitá pravidla a implementovat mechanizmy, které zajistí možnost změny co nejvíce parametrů bez nutnosti zásahu do kódu, popřípadě takový zásah zjednoduší. Nejnutnější předpokládané mechanizmy jsou:

- Objektově orientovaný návrh softwaru.
- Možnost volby oblastí kde se vyhledávají objekty uživatelem – oblasti zájmu.
- Možnost podrobnějšího nastavení pomocí konfiguračních souborů nebo specializované aplikace.
- Vedení záznamů o chování aplikace, stroje a jednotlivých fází výroby.
- Robustnost algoritmů pro zpracování obrazu.
- Jednoduchost ovládání i pro méně zkušenou obsluhu.
- Spolehlivost a přesnost (vzhledem k parametrům snímacího zařízení a zpracování v desítkách μm).

Objektově orientovaný návrh zlepší přehlednost programu, usnadní orientaci a respektuje skutečné členění jednotlivých částí řešené úlohy.([3],str. 77) Pokud budou jednotlivé třídy implementovány správně, usnadní tento přístup v budoucnu změny v řídícím softwaru. Pokud dojde ke změně hardwaru, například desky řídící servomechanizmy, lze snadno změnit postupy ovládání a řízení uvnitř třídy řídící

servomechanizmy, zatímco volání metod třídy zůstane stejné. Potom s sebou tato změna nese pouze nutnost změny vnitřní funkce jedné třídy bez nutnosti zásahu do zbytku aplikace.

Oblasti zájmu umožňují volbu nebo změnu jednotlivých oblastí obrazu pro zpracování. Navíc tuto změnu bude schopna provést i obsluha bez hlubší znalosti programu. Pokud dojde ke změně rozložení scény, nebo ke změně polohy některých dílů výrobku, je změna snadno proveditelná. Možnost určení oblastí a orientace scény navíc sníží nároky na zpracování obrazu z hlediska vyhledávání objektů a jejich orientace. Díky této dodatečné informaci může být zpracování obrazu rychlejší, jednodušší a robustnější.

Většina algoritmů, ať už vytvořených v rámci projektu nebo těch které jsou součástí OpenCV a dalších knihoven, přebírá často některé vstupní parametry, které by bylo dobré snadno měnit při odlaďování aplikace nebo při některých změnách na zařízení. Stejně tak některá další nastavení, ke kterým by neměl mít přímý přístup uživatel, a konfigurační nastavení by měla být dostupná zvenčí, bez nutnosti zásahu do kódu a nové kompilace. Tato nastavení mohou být uchovávána a přístupna pomocí konfiguračních souborů. Tento přístup ulehčí práci v případě změn, které jdou hlouběji než uživatelské změny, ale nevyžadují zásahy do kódu.

Vedení souborů s logy o funkci, chybách a výsledcích zpracování obrazu zjednoduší práci při vývoji aplikace a hledání chyb v návrhu. Později tyto záznamy umožní kontrolovat funkci stroje během provozu, nebo vytvářet statistiky výroby.

Zpracování obrazu by mělo vycházet z vlastností scény a mělo by využívat stálosti tvaru objektů a jejich uspořádání. Musí být však zachována určitá míra flexibility, aby nedošlo k přílišné specializaci a z toho plynoucím problémům v případě nutnosti změn v algoritmech. Zpracování obrazu by nemělo být závislé na velikosti obrazu a objektů, pokud ano, měla by být změna závislých parametrů snadno proveditelná. Pokud by došlo například ke změně kamery (jiné rozlišení) nebo objektivu (větší makro), není přijatelné, aby bylo nutné upravit algoritmy zpracování obrazu. Taková úprava je při větším množství algoritmů zdlouhavá a neefektivní.

3.2 SOFTWAROVÝ NÁVRH

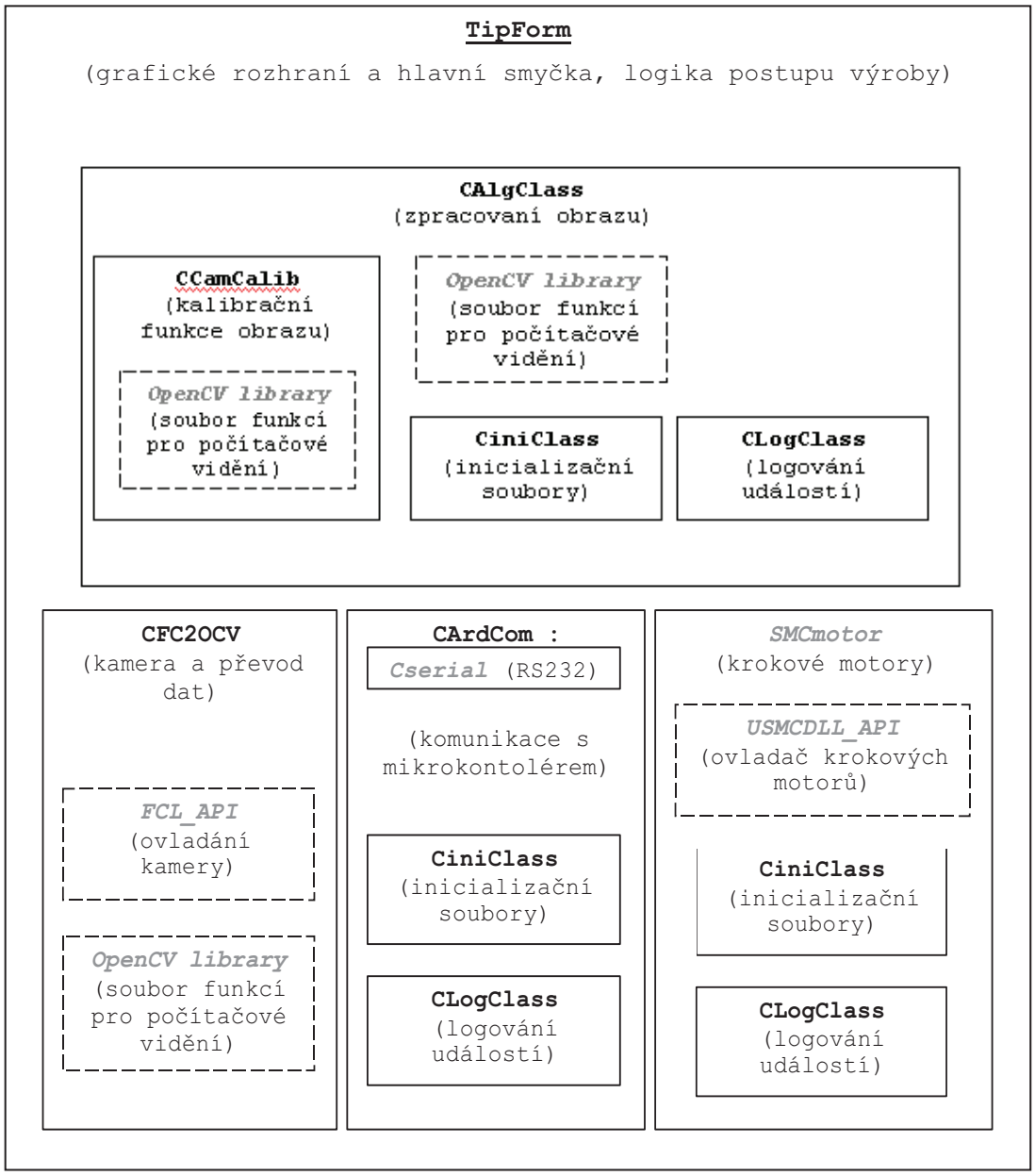
Celá architektura vytvořeného řídícího softwaru je vystavěna objektově. Jsou implementovány pomocné objekty, objekty pro ovládání hardware a objekty pro zpracování obrazu. Funkce každého objektu je specifická, kromě některých případů není použito dědění. Program je vystavěn jako pyramida. Objekty které jsou v pyramidě výše, využívají služeb nižších objektů. Vyšší objekty se nezajímají, jakým způsobem je operace provedena, zajímá je pouze výsledek – provedení operace nebo chyba.

Jako programovací jazyk byl vybrán jazyk C++, umožňuje zpětnou využitelnost funkcí v jazyce C a jedná se o nejčastější jazyk používaný při zpracování obrazu.([3],str. 77)

Některé objekty použité v programu zapouzdřují funkce v jazyce C do tříd. Těmito funkcemi jsou volání knihoven pro řízení hardware (kamera, krokové motory) a volání knihoven pro zpracování dat a dalších funkcí (ovládání sériové linky, zpracování obrazu prostřednictvím knihovny OpenCV). Zapouzdření do objektů je výhodné, zjednodušuje se implementace funkcí na vyšších vrstvách, úpravy v kódu jsou jednodušší a transparentní a nedochází k porušení koncepce objektově orientovaného návrhu. Dobrým příkladem je například třída starající se o snímací zařízení. Pokud má být získán obraz z použité kamery, je nutno provést řadu operací. Těmito operacemi jsou například: inicializace datových struktur, vyplnění inicializačních informací kamery, inicializace kamery, získání snímku voláním příslušné funkce s příslušnými parametry, přeformátování dat do struktury, kterou používá OpenCV knihovna… V každém kroku je také nutno kontrolovat a ošetřovat možné chyby. Pokud jsou tyto operace zapouzdřeny do objektu, který bude poskytovat službu (metodu) DejSnimek(), pak pro objekt využívající tuto službu je vše transparentní, voláním metody získá snímek, nebo dojde k chybě. Z hlediska vyššího objektu není nutno se o nic dalšího starat. Díky tomu dochází ke zjednodušení při programování vyšších vrstev.

Na Obr. 3 je schematicky zobrazena celá architektura vytvořeného software. Uzavřené obdélníky reprezentují objekty komunikující s okolím pomocí metod, obdélníky s čárkovaným ohraničením jsou funkce v jazyce C. Ne všechny části

software jsou prací autora, tyto jsou označeny *kurzívou*. Popis základních poskytovaných služeb objektů následuje dále.



Obr. 3: Architektura programu

V rámci práce na softwarové části projektu byly implementovány jednotlivé objekty, proveden celkový návrh a implementace výstavby aplikace je její logiky. Byl také proveden návrh postupu zpracování obrazu v jednotlivých fázích výroby,

jeho implementace s pomocí vhodných funkcí z knihovny OpenCV a otestování návrhu v provozních podmínkách.

3.3 POPIS OBJEKTŮ A NĚKTERÝCH METOD

Kompletní a detailní popis všech metod objektů pro zpracování obrazu a kalibraci kamery, jejich parametrů a návratových hodnot, stejně jako implementace je uveden v příloze práce (generováno pomocí programu DoxyGen). Na žádost zadavatele je dokumentace v anglickém jazyce. Kompletní dokumentace všech zdrojových kódů vytvořených v rámci práce je, z důvodu velkého rozsahu, přiložena na doprovodném datovém nosiči. Zde je uveden základní popis objektů a jejich metod nutný pro pochopení funkčnosti aplikace a jejího návrhu.

3.3.1 CIniClass

Třída pro práci se soubory obsahujícími nastavení aplikace. Konfigurační soubor není centrální, ale každá instance třídy používá vlastní soubor, jehož jméno je předáno při inicializaci třídy.

Soubor s nastavením je členěn do sektorů a položek. Sektor začíná jménem sektoru uzavřeným do hranatých závorek. Položka je tvořená názvem položky a hodnotou. Oddělovacím znakem je „=“. Znak středníku je vyhrazen pro komentáře.

Třída implementuje metody pro vytvoření sektoru, zápis a čtení položky ze souboru. Jsou podporovány tyto typy položek:

- Logická hodnota.
- Celočíselná hodnota.
- Hodnota s plovoucí desetinou čárkou.
- Řetězec znaků.

Příklad části inicializačního souboru:

```
[MAIN]
    SmoothSize=9
    ;komentář k hodnotě
    FileName=Pict.jpg
    Units=10.0
```

3.3.2 CLogClass

Třída určená pro zápis událostí v aplikaci do souboru. Cílový soubor je opět samostatný pro každý objekt a určený při inicializaci třídy. Typy záznamu jsou logicky členěny do skupin jako data, chyba, varování, start/konec aplikace a další. Jednotlivé položky mohou být ze záznamu vypuštěny v závislosti na nastavení. Metoda zápisu vytvoří zápis do souboru v závislosti na typu události a předaném řetězci, který má být uložen. Položka záznamu v souboru se skládá z typu události, časového razítka a řetězce, který byl předán metodě pro zápis.

Vedení záznamů o chování aplikace umožňuje vysledovat nejenom chybu, která je hlášena aplikací, ale také příčiny jejího vzniku. Tento přístup zrychluje odlaďování aplikace při vývoji a zlehčuje hledání chyb v pozdějších fázích vývoje a během provozu.

3.3.3 CFC2OCV

Výrobce snímacího zařízení, společnost FCLab, poskytuje uživatelské rozhraní a knihovny v jazyce C. Jak již bylo uvedeno výše na příkladu výhodnosti zapouzdření, je knihovna dobře použitelná, ale z hlediska vyšší třídy je její použití příliš komplikované. Tato třída poskytuje mnoho přetížených metod (využití polymorfismu jazyka C++) jež vrací volajícímu objektu obraz ve formátu který je využívaný knihovnou OpenCV. Při volání je možno zvolit jakýkoliv parametr obrazu poskytovaný kamerou – rozlišení, offset, ROI, dobu expozice a další. Třída obstarává všechnu nutnou práci na pozadí, inicializaci všech datových struktur, inicializaci a nastavení parametrů kamery, ošetření chyb.

Součástí třídy jsou i metody pro spuštění a ovládání snímací smyčky, kdy kamera kontinuálně snímá jeden snímek za druhým. Rychlost snímání je omezená pouze rychlostí zařízení. Při volání metody pro získání snímku je pak získán naposledy zachycený snímek prakticky v nulovém čase, není nutné čekat na expozici. Jedná se vlastně pouze o kopírování posledního snímku. Snímací smyčka je implementována jako samostatné vlákno, běžící v nekonečné smyčce. K ovládání

vlákna a předávání dat je využito synchronizačních objektů poskytovaných operačním systémem Windows.

3.3.4 CArdCom

Tato třída obstarává komunikaci s mikrokontrolérem řídícím další HW, jako jsou servomotory, elektronické spínače ventilů, relé, solenoidy a další. Komunikace pracuje na fyzickém rozhraní RS232. Pro jednoduché ovládání sériové linky je využito třídy CSerial, jejímž autorem je Ramon de Klein. Třída pro ovládání sériové linky je distribuována pod GNU Lesser General Public License, což znamená, že kód je zdarma, bez omezení, ale bez záruky použitelný pro jakoukoliv, i komerční, aplikaci.

Komunikace probíhá pomocí zasílání zpráv. Zpráva je formátována v ASCII kódu, je tedy čitelná člověkem, což sice zlepšuje přehlednost, ale zároveň zvětšuje délku zprávy. Objem zasílaných dat je však malý a jejich větší velikost zde nevadí. Součástí zprávy je kontrolní součet pro kontrolu chyb. Reakcí kontroléru je provedení požadovaného úkonu a zaslání ACK, nebo naopak NCK. Jedná se o komunikační model Master(PC) / Slave(µC). Komunikaci inicializuje vždy PC a buďto vyžádá data, nebo zašle příkaz.

Třída poskytuje metody k přímému ovládání hardwaru. Obstarává inicializaci sériové linky, sestavování a zasílání zpráv a kontrolu jejich provedení a potvrzení, stejně jako získávání dat ze zařízení. Uživateli objektu jsou poskytnuty metody pro inicializaci zařízení, ovládání jednotlivých kusů hardware (servomotory, spínače, ventily, relé) a získávání dat (měřící rezistory, stavy pinů). Pro uživatele objektu této třídy je jeho použití opět velmi jednoduché a transparentní. Vnitřní funkčnost třídy je přímo spjata s implementací software v mikrokontroléru a se zapojením hardwaru ovládaným kontrolérem. Autorem programu pro mikrokontrolér a návrhu zapojení hardwaru je kolega Bc. Tomáš Hynčica. ([5])

3.3.5 SMCmotor

Ovládání krokových motorů je koncipováno stejným způsobem, jako ovládání kamery. Výrobce opět poskytuje ovládací funkce v jazyce C a tyto funkce jsou zapouzdřeny do třídy. Poskytované metody umožňují inicializaci, relativní i absolutní pohyb, kalibraci krajových poloh a další funkce. Autorem třídy rozhraní pro krokové motory je kolega Bc. Tomáš Hynčica. ([5])

Kromě funkce zapouzdření a poskytnutí metod pro jednoduché ovládání krokových motorů, poskytuje třída i rozšířenou funkčnost. Toto rozšíření je v podobě metod, určených pro vytvoření posloupnosti bodů cesty a možnosti mezi těmito body procházet. Posloupnost bodů spolu s informací o typu bodu je uložena v konfiguračním souboru a je prostřednictvím třídy využitelná i modifikovatelná. Uložená cesta je určena pro zaznamenání jednotlivých montážních bodů a jejich vzájemné návaznosti. Prostřednictvím třídy je tak možno snadno přecházet mezi jednotlivými stanovišti montážní linky, bez nutnosti starat se o ovládání krokových motorů.

Pohyb mezi jednotlivými stanovišti, stejně jako například kalibrace krajních bodů, je časově náročný. Funkce konající pohyb čeká ve smyčce na jeho dokončení. Během tohoto času je nutno zajistit funkčnost řídící aplikace a její ovladatelnost. Proto je opět použito vícevláknové řešení. Voláním metody pro pohyb je spuštěno vlákno obstarávající pohyb a návrat z volané metody je téměř okamžitý. Třída nedovolí volání další metody, dojde k vrácení příslušné chyby, dokud nedojde k dokončení pohybu. Výjimkou jsou specifické metody jako zastavení, zjištění stavu operace a podobné. Toto řešení umožňuje čekat na dokončení ve vyšší vrstvě, nebo ve stejné době vykonávat jinou činnost.

3.3.6 CCamCalib

Při pořizování obrazu kamerou s objektivem s vysokým zvětšením dochází, vlivem optiky objektivu, ke zkreslení obrazu. V tomto případě navíc není zaručeno, že je obraz snímán kamerou umístěnou kolmo ke snímané scéně. Při snímání dochází

ke zkreslení vlivem nesouběžnosti roviny scény se snímací rovinou kamery a ke zkreslením vlivem objektivu.

Kalibrační třída poskytuje metody pro korekci těchto zkreslení. Korekce vychází ze zjištění funkce zkreslení pomocí kalibračního obrazce. Jsou zde využity kalibrační funkce z knihovny OpenCV, podrobný popis je uveden v textu dále v kapitole 4.1 zabývající se zpracováním obrazu. Třída poskytuje metody pro zjištění zkreslení kamery, korekci tohoto zkreslení a metody pro převod mezi počtem pixelů a vzdáleností v jednotkách SI. Přesný přepočet mezi počtem pixelů a vzdáleností je kritický pro správné určení polohy jednotlivých částí nosníku při montáži. Princip zjištění vzdálenosti a přepočtu je opět uveden dále v kapitole zabývající se zpracováním obrazu.

3.3.7 CAlgClass

Většina montáže je řízena kamerou na základě dat získaných pomocí zpracování obrazu. Tato třída sdružuje všechny nutné postupy zpracování obrazu k získání informací o stavu montáže. Opět je využito funkcí knihovny OpenCV. Jedná se o nejrozsáhlejší třídu v projektu. Jednotlivé použité postupy jsou podrobně zpracovány v kapitole o zpracování obrazu.

Třída poskytuje metody pracující s pořízeným obrazem scény. Výstupem zpracování je ve většině případů vzdálenost mezi objekty nebo poloha jednotlivých objektů v obrazu. Typ zpracování závisí na fázi montáže a na typu prováděného úkonu. Funkčnost třídy je přímo vázána na vzhled scény a technické řešení celého stroje, ačkoliv, díky kalibraci obrazu a použití inicializačních souborů, připouští určitou omezenou volnost parametrů.

3.3.8 TipForm

Je třídou vzniklou při vytvoření CLR (Common Language Runtime) projektu v MS Visual Studiu. Není nutnou součástí pro funkčnost projektu, ale tento přístup byl zvolen pro jednoduchou implementaci uživatelského rozhraní. Třída obstarává

zprávu objektů grafického rozhraní a akcí při jejich aktivaci. Grafické rozhraní usnadňuje práci obsluze zařízení a zjednodušuje orientaci v programu.

Během vývoje jsou součástí této třídy také postupy zpracování jednotlivých fází montáže. Umístění řízení výroby do této třídy odporuje dosavadní logice výstavby aplikace. Tento rozpor vznikl během testování jednotlivých kroků a odlaďování, kdy se jednalo o nejpřímější cestu, jak otestovat funkčnost jednotlivých částí aplikace. V budoucnu by mělo dojít k rozdělení na dva samostatné objekty, z nichž jeden bude obstarávat pouze grafické rozhraní a zasílání zpráv o akcích uživatele, druhý se bude starat o správu montáže a provedení jednotlivých montážních kroků. Toto rozdělení umožní logické oddělení celého výrobního procesu od grafického rozhraní a nezávislost všech funkčních částí aplikace na použitém grafickém rozhraní a typu jeho implementace.

3.4 VÍCEVLÁKNOVÉ ŘEŠENÍ NĚKTERÝCH METOD

Pokud se podíváme na postup výroby hrotu (kalibrace, vyzvednutí nosníku, nanesení lepidla,…) pak tento postup přímo vybízí k realizaci aplikace pracující v supersmyčce, kdy jednotlivé kroky montáže jsou vykonávány sekvenčně. Aplikace je skutečně realizována jako supersmyčka. Některé výrobní postupy jsou časově náročné, například vytvrzení lepidla trvá několik minut. Tato časová náročnost při zpracování v supersmyčce způsobí při čekání nemožnost jakéhokoliv ovládání aplikace a je nutné počkat na dokončení akce. Tato vlastnost je nepříjemná hlavně z hlediska obsluhy pracující se zařízením. Z důvodu lepší ovladatelnosti programu a zařízení byly proto časově náročné akce implementovány jako samostatné vlákno. Voláním příslušné metody dojde ke spuštění smyčky, která akci vykoná a návrat z metody je proveden po spuštění příslušného vlákna.

Je nutné si uvědomit, že oproti použití implementačně jednoduché smyčky klade paralelní vícevláknový návrh mnohem větší nároky na správnou implementaci. Je nutné správně implementovat synchronizaci mezi jednotlivými vlákny a omezit přístup k jednotlivým sdíleným zdrojům.

V rámci projektu jsou vícevláknová řešení použita v objektech CFC2OCV a SMCmotor. Další objekt, který by bylo možno implementovat vícevláknově, je

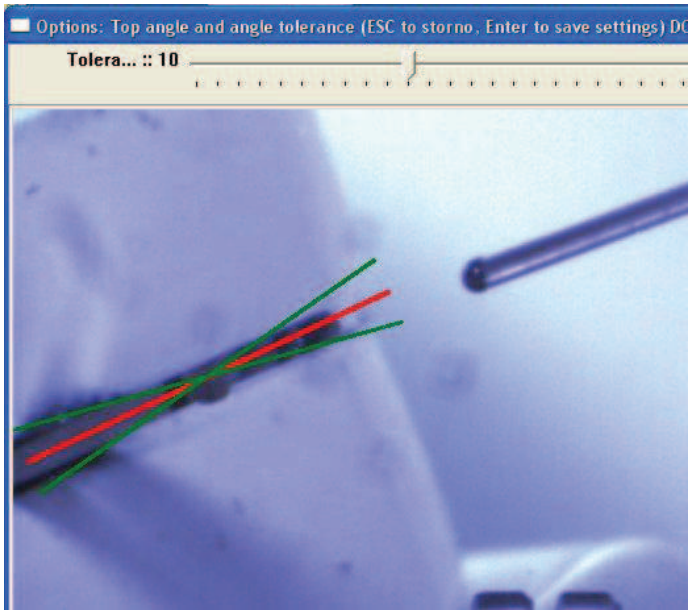
komunikační třída CArdCom. Avšak pouze v případě, že by došlo ke změně hierarchie a mikrokontroléru by bylo dovoleno také aktivně navazovat komunikaci s PC. Také ve třídě realizující posloupnost a logiku montáže, kdy by ovládání procesu bylo realizováno zasíláním zpráv mezi dvěma vlákny (grafické rozhraní – montážní vlákno), by nalezl paralelismus svoje místo.

Objektově orientovaný návrh v tomto bodě usnadňuje správu sdílených zdrojů. Ke každé části stroje je přistupováno jako k samostatnému objektu, ať už se jedná o použitý software nebo hardware. Každý objekt má proto možnost spravovat přístup ke svým zdrojům a správa zdrojů je ponechána pouze na něm. Toto řešení předpokládá, že ke sdíleným zdrojům není nikdy možno přistupovat jinak, než prostřednictvím příslušného objektu. Samozřejmě z tohoto důvodu musí být splněno pravidlo, že instance každého objektu, pokud pracuje paralelně a k jeho zdrojům musí být zajištěn bezpečný přístup, se může v aplikaci objevit pouze jednou. V jiném případě by bylo nutno zajistit náležité sdílení zdrojů mezi instancemi stejného objektu.

Při tvorbě vláken v jednotlivých objektech byly využity služby poskytnuté operačním systémem Windows pomocí volání systémových funkcí. Stejně tak pro synchronizaci a přístup ke sdíleným zdrojům a kritickým sekcím byli využity synchronizační objekty poskytované tímto operačním systémem.

3.5 NASTAVENÍ PARAMETRŮ ZPRACOVÁNÍ OBRAZU

Zpracování obrazu potřebuje poměrně velké množství nastavení jednotlivých algoritmů. Jedná se jak o parametry buďto kritické pro funkčnost (nastavení prahů, cannyho detektoru, Houghovy transformace,…), nebo parametry zjednodušující zpracování a zvyšující robustnost (oblasti zájmu, předpokládaná orientace objektů a tolerance hodnot,…). Všechny parametry jsou uloženy v inicializačních souborech aplikace. Přímá změna však vyžaduje znalost implementace a je pro uživatele nepohodlná. Z tohoto důvodu je součástí aplikace možnost provést jednotlivá nastavení v okamžiku, kdy probíhá konkrétní zpracování.



Obr. 4: Příklad interaktivního nastavení parametrů zpracování obrazu

Uživateli je nabídnuto okno s jednoduchou možností nastavení pomocí myši (oblasti zájmu,…) nebo pomocí grafických prvků (prahy,…). Provedené změny jsou ihned patrné, aby bylo možno zhodnotit jejich dopad. Jednotlivá nastavení jsou po potvrzení uložena do inicializačního souboru. K nastavení jsou nabídnuta pouze nejdůležitější nastavení algoritmů. Podrobnější nastavení jsou mimo možnosti uživatele a jejich změna není nutná, pokud nedošlo ke změně scény. Taková úprava však vyžaduje hlubší odborný zásah.

Na Obr. 4 je příklad nastavení úhlu osy nosníku pomocí myši a tolerance daného parametru pomocí posuvníku. Nastavení je pro uživatele přímo zobrazeno.

3.6 SHRNUTÍ NÁVRHU APLIKACE

Aplikace je vystavěna objektově, na principu komunikace mezi jednotlivými objekty, probíhající pomocí metod těchto objektů. Jednotlivé objekty reprezentují rozdělení zpracování specifických úloh pomocí softwaru nebo reprezentují hardware. Návrh respektuje logiku řešení úlohy a jeho rozdělení na samostatné části.([3],str. 77) Celý proces výroby je řízen v supersmyčce, která se však, v některých místech a při určitých typech operací, štěpí do více vláken.

Architektura aplikace je částečně výsledkem původního záměru na počátku zpracování projektu, částečně výsledkem poznatků během implementace. Není možné navrhnout ihned na počátku pevné schéma, které se během práce nezmění a bude vyhovovat všem požadavkům a úpravám. Jediným pevným použitým schématem je v tomto případě požadavek na objektově orientovaný návrh, vedoucí k lepší transparentnosti aplikace.

Samozřejmě mohou existovat připomínky k jednotlivým částem aplikace a k členění objektů. Inicializační soubory a logy by mohly být centrální a přístup k nim řízen samostatným objektem, místo současného řešení, kdy vlastní každý objekt vlastní soubor a instanci odpovídajícího objektu. Metody pro správu a využití uložené cesty krokových motorů po výrobní lince by mohly být vyčleněny do samostatné třídy. Došlo by tak k oddělení ovládání motorů od správy posloupnosti výrobních bodů. Grafické rozhraní by mělo být odděleno od logiky výroby. Určitě by se našlo i mnoho dalších připomínek nebo alternativních řešení. Architektura použitá v projektu je pouze jednou z mnoha možností, přináší své výhody i nevýhody, stejně tak jako mají své výhody i nevýhody alternativní přístupy k řešení.

4. ZPRACOVÁNÍ OBRAZU

Při montáži nosníku je nutno postupovat v několika technologických krocích. Při většině úloh v tomto procesu je nutné zpracování obrazu, aby byla získána zpětná vazba pro řízení a také data pro kontrolu kvality během montáže. Jednotlivé kroky jsou: nanesení lepidla, montáž hrotu, ostření hrotu leptáním. V následujících kapitolách jsou popsány postupy zpracování obrazu pro tyto kroky.

Výhodou úlohy je skutečnost, že se jedná o průmyslové zpracování obrazu. Některé objekty ve scéně mají stálou a předem definovanou pozici a stejně tak se nepředpokládá změna uspořádání nebo osvětlení. Avšak vyskytují se také problémy jako malá hloubka ostrosti, složité scény (leptání), odlesky,…

Na tomto místě je nutno také uvést problémy s nasvětlením scény během realizace. Prvotní experimentování v laboratorních podmínkách vedlo cestou nasvětlení scény takovým způsobem, aby objekty byly světlé oproti tmavému pozadí. Později se ukázalo jako problematické zajistit stálý odraz vzhledem k vlastnostem objektů a prostorovému řešení stroje. V pozdější fázi bylo voleno bílé pozadí a nasvícení odzadu, objekty jsou poté tmavé proti pozadí. Při použití bílého pozadí stačí osvětlení difúzním světlem, narozdíl od bodového osvětlení vázaného na polohu kamery při použití tmavého pozadí. Bez ohledu na použité osvětlení, všechny metody zpracování obrazu využívají kontrastu mezi objektem a jeho pozadím. Typy použitého nasvětlení jsou z hlediska algoritmů ekvivalentní a algoritmy jsou shodné.

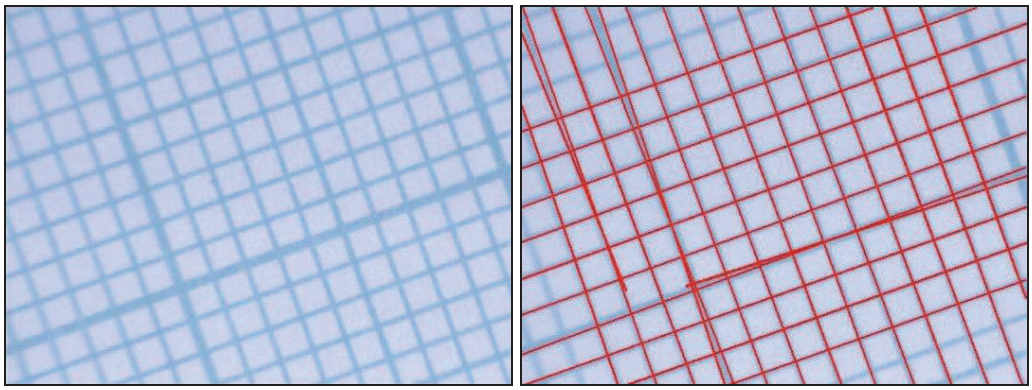
Všechny zde popsané algoritmy jsou součástí třídy CAlgClass nebo třídy CCamCalib.

4.1 KALIBRACE VZDÁLENOSTI A KOREKCE ZKRESLENÍ

Krokové motory, použité pro polohování nosníku na montážní lince, mají definovanou vzdálenost posunutí na každý krok. Protože snímání a zpracování obrazu bude použito pro zjištění vzdáleností ve scéně, je nutné zjistit převodní konstantu pixel/vzdálenost.

Pro kalibraci je nutné použít obrazec o známých rozměrech a takovém uspořádání, aby bylo možné určit vzdálenosti a popřípadě natočení scény. Jednou z možností je použití obrazu, který obsahuje přímky a Houghovu transformaci.

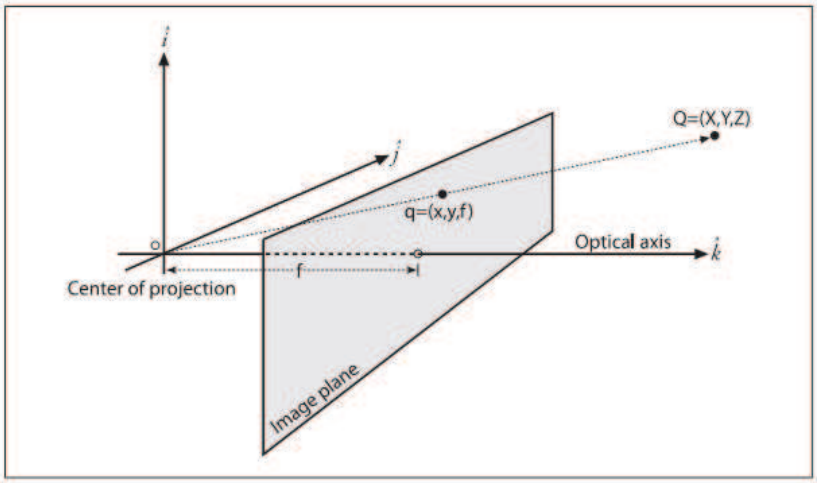
Na Obr. 5 je příklad snímku milimetrového papíru a poté aplikované Houghovy transformace. Takovýto obrazec není příliš vhodný z důvodu velké šířky čar, kdy dochází ke vzniku zkreslení vlivem jejich nenulové tloušťky. Ideálnějším obrazcem je například šachovnice. Hrana mezi světlými a tmavými poli má ideálně nulovou šířku, nebo jednotkovou, pokud nezanedbáme chybu vzniklou vzorkováním obrazu do snímací mřížky čipu kamery. V obrazci šachovnice lze využít detekce významných bodů v podobě rohů jednotlivých polí šachovnice.(Obr. 7) Právě šachovnice byla použita pro vyřešení problému korekce zkreslení i zjištění převodní konstanty pro převod mezi počtem bodů v obrazu a vzdáleností ve snímané scéně.



Obr. 5: Výsledek HT aplikované na snímaný obrazec čar

4.1.1 Korekce zkreslení kamery

Pokud je zanedbán vliv čočky, je možno kameru modelovat jednoduchým způsobem. Nejjednodušeji průchodem světla přes malý otvor a projekcí na projekční rovinu. Pokud je zaměněna projekční rovina s otvorem, je možno snímání obrazu znázornit tak, jak je uvedeno na Obr. 6.



Obr. 6: Projekce objektu na projekční plochu jednoduché kamery [1]

Projekci kamery pak můžeme jednoduše zapsat dvojicí rovnic [1]:

$$x_{obr} = f_x\left(\frac{X}{Z}\right) + c_x \qquad y_{obr} = f_y\left(\frac{Y}{Z}\right) + c_y \tag{1}$$

Kde f_x a f_y jsou ohniskové vzdálenosti v obou osách a c_x,c_y posunutí projekční roviny v těchto osách.

Projekce, která mapuje bod $Q_i(X_i,Y_i,Z_i)$ z fyzického světa na bod na projekční ploše čipu snímače, se nazývá projekční transformace. Při použití projekčních transformací většinou pracujeme s homogenním souřadným systémem. V tom případě přechází vektor z rozměru n do rozměru $n+1$. Platí pravidlo, že každé dva body, jejichž koordináty jsou proporcionální, jsou shodné. Projekční plocha čipu je dvourozměrná. Platí pro ni tedy souřadný systém $q(x,y,w)$. Za předpokladu, že dva body jsou ekvivalentní, pokud jsou proporcionální, je možno zjistit polohu bodů podělením hodnotou f. Díky těmto vlastnostem je možno sestavit matici vnitřních parametrů kamery (f_x,f_y,c_x,c_y) do matice M. Projekce bodu na projekční plochu kamery a matice vnitřních parametrů kamery [1]:

$$q = MQ, \qquad \text{kde} \qquad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \; M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \; Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2}$$

Každá čočka, kromě ideální, způsobuje deformaci obrazu. Dvě hlavní deformace obrazu jsou radiální a tangenciální.

Radiální zkreslení vzniká v oblastech okraje obrazu a způsobuje tzv. efekt rybího oka. Toto zkreslení je důsledkem tvaru čočky. Velikost zkreslení roste směrem od středu obrazu. Radiální zkreslení lze korigovat za znalosti všech parametrů k_i a r, dvojicí rovnic (3)[1], kde koeficienty r^l jsou rozvojem sudých koeficientů Taylorovi řady funkce zkreslení směrem od optického středu obrazu ($r=0$):

$$\begin{aligned} x_{kor} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{kor} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \qquad (3)$$

Tangenciální zkreslení vzniká chybou při výrobě kamery. Čočka není v tomto případě přesně rovnoběžná s čipem kamery. Pro korekci tangenciálního zkreslení je nutný čtvrtý a pátý parametr p_j. [1]

$$\begin{aligned} x_{kor} &= x + \left[2p_1 y + p_2(r^2 + 2x^2)\right] \\ y_{kor} &= y + \left[2p_2 x + p_1(r^2 + 2y^2)\right] \end{aligned} \qquad (4)$$

Samozřejmě existují další typy zkreslení optické soustavy. Většinou jsou však méně významné. Kalibrační funkce v knihovně OpenCV jsou schopny vypočíst a používat korekční vektor o uvedených pěti parametrech.

Pro plnou kalibraci kamery v OpenCV je nutno zjistit 10 parametrů (4 vnitřní + 3 rotace a 3 translace kalibračního vzoru) pro výpočet projekční matice H a 5 parametrů pro korekci optiky kamery – k_1, k_2, k_3, p_1, p_2.

Při aplikaci korekční matice se vychází z projekčního mapování. Projekční mapování lze popsat jako transformaci dvourozměrné plochy na jinou dvourozměrnou plochu. Projekci dvourozměrného povrchu na projekční plochu kamery je možno zapsat rovnicí [1]:

$$\tilde{q} = sMW\tilde{Q} \qquad (5)$$

$$\tilde{Q} = [X \quad Y \quad Z \quad 1]^T$$

$$\tilde{q} = [x \quad y \quad 1]^T$$

$W = [R \quad t]$ fyzické parametry translace a rotace

s změna měřítka

Matematickou úpravou [1]

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM[r_1 \quad r_2 \quad r_3 \quad t] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sM[r_1 \quad r_2 \quad t] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{6}$$

Lze přepsat rovnici do tvaru

$$\tilde{q} = sH\tilde{Q} \tag{7}$$

Kde H je projekční matice kamery o rozměru 3x3. Souřadnice mezi bodem na snímané ploše a bodem projekční plochy lze přepočítat pomocí rovnic [1]:

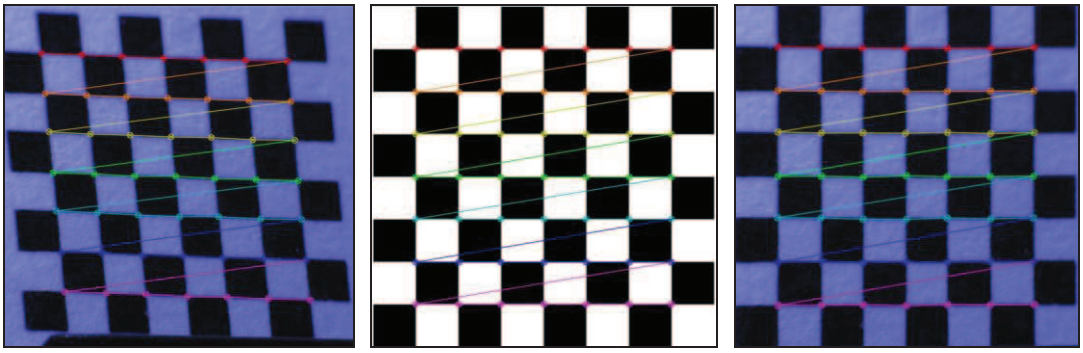
$$p_{cíl} = Hp_{zdroj}, \qquad p_{zdroj} = H^{-1}p_{cíl} \tag{8}$$

$$p_{cíl} = \begin{bmatrix} x_{cíl} \\ y_{cíl} \\ 1 \end{bmatrix}, \qquad p_{zdroj} = \begin{bmatrix} x_{zdroj} \\ y_{zdroj} \\ 1 \end{bmatrix} \tag{9}$$

OpenCV používá výše uvedený matematický aparát a matici H pro transformace spojené s kalibrací obrazu. Matice H spolu s vektorem distorzních koeficientů čočky a dalšími parametry je výstupem kalibrační funkce `cvCameraCalibration2()`. Vstupem funkce jsou vektory korespondujících bodů předlohy a téže předlohy snímané kamerou. Kalibrační funkci teoreticky postačuje pro výpočet trojice bodů. S použitím většího množství bodů však roste robustnost a přesnost výpočtu. Pro bližší informace o matematickém aparátu použitém při výpočtu kalibrační matice a parametrů v knihovně OpenCV viz zdroj ([1], kapitola 11).

Pro výpočet transformační matice v projektu byl použit obraz šachovnice. Pro získání významných bodů byla využita poskytovaná funkce `cvFindChessboardCorners()`. Funkce umožňuje nalezení rohových bodů polí

šachovnice libovolných rozměrů. Tyto body jsou poté využity jako vstup do kalibrační funkce. Obecně poslouží jakékoliv vektory korespondujících bodů o dostatečné délce. Je tedy možno využít i vlastní implementace s vlastním obrazcem libovolného vzhledu.



Obr. 7: Významné body snímku a jeho předlohy a výsledek aplikace korekční matice

Knihovna OpenCV poskytuje také funkce pro aplikaci transformační matice na obraz v přímém i inverzním směru a pro korekce zkreslení optické soustavy. Třída CCamCalib v této úloze obstarává rozhraní pro výpočet transformační matice a korekčních koeficientů, jejich uložení a transformaci obrazu.

4.1.2 Kalibrace měřítka obrazu

Scéna s již aplikovatelnou korekcí je snadno použitelná pro výpočet převodní konstanty. Počet polí šachovnice je znám. Počet bodů obrazu je dán rozlišením kamery. Jedinou další informací, jejíž znalost je potřebná pro výpočet převodní konstanty, je rozměr použité předlohy šachovnice. Pokud jsou známy rozměry kalibrační šachovnice, je možno určit převodní konstantu. Konstanta pro převod pixel-délka:

$$k = \frac{N_p}{d} \qquad [pixel/mm] \qquad (10)$$

N_p – počet pixelů v daném směru

d – délka šachovnice v předloze v daném směru

Výpočet je přímý díky tomu, že předchozí korekce zkreslení nejenže odstranila nepravidelnosti obrazu, ale také došlo k transformaci obrazu takovým způsobem, že kalibrační obrazec přesně vyplňuje obraz získaný po aplikaci korekcí. Výpočet převodní konstanty je pak velice jednoduchý. Informace o přesných rozměrech šachovnice je kalibrační třídě předána pomocí inicializačního souboru třídy zpracování obrazu.

Zjištěnou vzdálenost v obrazu je pak možno, díky znalosti převodní konstanty, snadno přepočítat na vzdálenost v metrických jednotkách, kdykoliv je to během zpracování obrazu potřeba. Velikost měřítka je závislá na nastavení objektivu a rozlišení obrazu. Při praktických testech typicky 10µm na jeden bod obrazu při rozlišení 800×800 bodů. Při plném rozlišení a makru je však možno dosáhnout lepší přesnosti než 5µm na bod obrazu.

4.2 PŘEDZPRACOVÁNÍ, SEGMENTACE A POPIS OBJEKTŮ

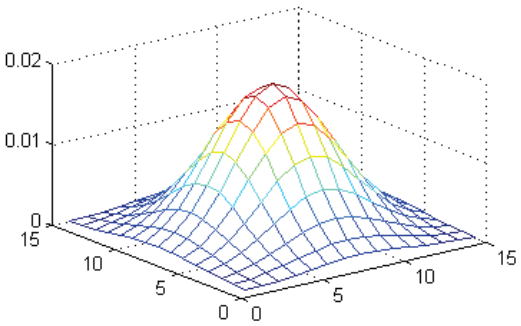
Pro zpracování obrazu v jednotlivých algoritmech je potřeba nejprve upravit vstupní data. Je nutné obraz předzpracovat, provést základní segmentaci a zpracování obrazu a objekty v obrazu vhodným způsobem popsat. Takto připravená data jsou poté využitelná následnými algoritmy.

4.2.1 Filtrace šumu

Použitý objektiv s velkým zvětšením má nízkou světelnost. Při snímání je třeba použít poměrně dlouhou expozici v řádech stovek milisekund. Dochází ke generaci šumu, který je nutno před dalším zpracováním z obrazu odstranit.

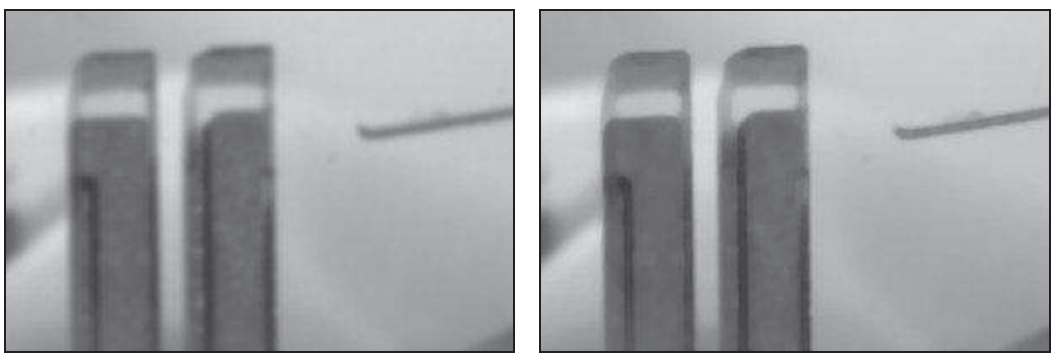
Jednou z možností filtrace je lokální průměrování s použitím konvoluční masky s 2D Gaussovým normálním rozložením koeficientů. V pozdějším zpracování se používá Cannyho hranový detektor, který se ukázal poměrně citlivý na šum při použití nižších prahů. Použitá konvoluční maska je proto poměrně velká, o délce hrany okolo 9 bodů při rozlišení obrazu 800×600 obrazových bodů. Gaussova křivka v 2D prostoru:

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (11)$$



Obr. 8: Gaussova konvoluční maska

Lokální filtrace Gaussovou konvoluční maskou dobře potlačuje šum, ale zároveň snižuje výraznost hran v obrazu. Následné zpracování založené na hranách je poté obtížné, zvláště u objektů mimo rovinu snímání, jejichž hrana je již narušená nízkou hloubkou ostrosti. Filtrace mediánem je v tomto případě výhodnější. Poskytuje obdobné výsledky jako lokální filtrace a neporušuje výrazně hrany objektů. Je však nutné dbát na to, aby velikost masky, její velikostní ekvivalent v μm, pro filtraci výrazně nepřekročila rozměr nejmenších objektů v obrazu. Došlo by k odfiltrování těchto objektů. Porovnání obou způsobů filtrace pro masku o rozměrech 9×9 (Gauss) a 7×7 (medián) obrazových bodů je na Obr. 9. Medián, jako výhodnější metoda, je filtrem šumu použitým v projektu.



Obr. 9: Porovnání filtrace Gaussovou konvoluční maskou a mediánem

4.2.2 Cannyho hranový detektor

Cannyho hranový detektor využívá první derivace obrazu ve směrech *x* a *y*. Tyto derivace jsou pak kombinovány do čtyřsměrné derivace.([1]) Lokální maxima této derivace jsou adepty na sestavení hrany. Při sestavování hrany se vychází z dvojice prahů pro prahování s hysterezí. Pokud má bod hodnotu gradientu nad úrovní vyššího prahu, je přijat jako hrana, gradient pod úrovní nižšího prahu je odmítnut. Body s hodnotou gradientu v rozmezí obou prahů jsou přijaty pouze v případě, že jsou spojeny s bodem nad úrovní vyššího prahu. Většinou se jako doporučený rozsah udává poměr prahů v rozmezí 2:1 až 3:1.([1])

Výsledkem Cannyho detektoru je binární obraz hranic objektů, které splňují výše uvedená kritéria. V pozdějším zpracování obrazu je, v rámci projektu, výstup Cannyho detektoru nejčastěji použit jako vstupní binární obraz pro aplikaci Houghovy transformace. Výsledek aplikace Cannyho detektoru na obraz snímané scény je na Obr. 13.

4.2.3 Adaptivní prahování

Prahování je nejstarší a nejjednodušší segmentační metoda, selhává však v případě nerovnoměrně nasvětlené scény. V projektu nelze zajistit dokonale homogenní nasvětlení pozadí. Limitujícími faktory jsou prostorové řešení stroje a pohyb kamery společně se snímaným objektem, zatímco pozadí a z větší části i osvětlení je stacionární. Světelné podmínky se proto během montáže mohou měnit.

Adaptivní prahování ([1]) je méně citlivé na nerovnoměrné nasvícení pozadí. Práh v tomto případě není jedna globální hodnota, ale je určen pro každý bod obrazu samostatně. Pro okolí bodu o velikosti *N* je určena průměrná hodnota jasu, používá se vážený průměr s váhami danými 2D Gaussovým rozložením v okolí prahovaného bodu. Tato průměrná hodnota je poté použita jako práh pro daný bod. Pro dobré výsledky je ještě nutno připočítat k průměrné hodnotě offset, který posouvá hodnotu prahu, omezí se tak například vliv šumu v obrazu.

Nevýhodou adaptivního prahu je omezení velikosti prahovaných objektů. Objekty blízké velikosti okolí bodu, stejně jako objekty podstatně menší nejsou

prahovány správně. Další nevýhodou je citlivost na hranice oblastí s rozdílným jasem.

Pro nalezení drátku a jehly v obrazu, jak bude uvedeno dále, je však tato segmentace dostačující. U ostatních objektů v obrazu bylo později od použití prahování upuštěno. Důvodem byla nedostatečná robustnost a hlavně přesnost zjištěné polohy a tvaru jednotlivých objektů. Výsledek segmentace adaptivním prahováním je na Obr. 13.

4.2.4 Barvení objektů

Prahovaný obraz je možno rozlišit na jednotlivé objekty barvením. Barvení znamená označení každého jednotlivého objektu v segmentovaném obrazu jedinečným indexem.([8], str. 452) Podle indexu lze separovat jednotlivé objekty a posléze s nimi pracovat samostatně. Knihovna OpenCV neobsahuje přímo funkci pro obarvení objektů. Implementace barvení je však dosti zlehčena jinými poskytovanými funkcemi. Po rozlišení jednotlivých objektů je nutno rozhodnout, které objekty a které jejich části jsou v obrazu zajímavé a kde se nacházejí.

Barvení bylo implementováno pomocí funkce `cvFloodFill()` poskytované knihovnou OpenCV. Obecně se používá k vyplnění ploch v obrazu, které mají podobné parametry.([1], str. 124) Funkce přiřadí stejnou barvu bodu a jeho libovolně velkému okolí, ve kterém je splněna nastavená podmínka nepřekročení maxima změny jasu mezi dvěma sousedními body. Jelikož objekt v binárním obrazu má pouze jednu barvu, dojde k zastavení jeho barvení na hranicích s pozadím. Postup barvení je následující:

1. Procházej celý obraz P(i,j).
2. Pokud narazíš na neoznačený objekt (P(i,j)=255) - obarvi ještě nepoužitou barvou (dojde k obarvení celé plochy objektu s přihlédnutím k osmi sousedům).
3. Opakuj bod 2, dokud neprojdeš celý obraz nebo nevyčerpáš jasový rozsah obrazu.

Na Obr. 10 je zobrazen výsledek barvení objektů. Funkce `cvFloodFill()` vrací strukturu obsahující plochu objektu a parametry obdélníku přesně vymezujícího

polohu a velikost objektu v obrazu. Po dokončení barvení je, s využitím získaných dat, obraz rozčleněn na seznam samostatných obrazů jednotlivých objektů a jejich parametrů. Obrazy, jejich barva a parametry jsou později využity k identifikaci těchto objektů.

Obr. 10: Výsledek barvení objektů (rozsah barev upraven pro větší názornost)

4.2.5 Popis objektů

Pro rozpoznání jednotlivých objektů je nutný jejich popis. V případě zpracovávané scény postačuje většinou jednoduchý popis. Byly využity:

1. Radiometrické deskriptory založené na regionech ([4],[8])
   - Velikost – počet pixelů plochy objektu.
   - Obvod – počet obvodových pixelů.
   - Podlouhlost – poměr stran opsaného obdélníku, zde však bez splnění podmínky nejmenšího obsahu.
2. Momenty a momentové invarianty ([1],[2],[4])
   - Těžiště objektu.
   - Orientace objektu.

Geometrický moment řádu $p+q$:

$$m_{pq} = \sum_{Y}\sum_{X} x^p y^q s(x,y) \qquad (12)$$

Výpočet těžiště a objektu:

$$x_t = \frac{\sum_{Y}\sum_{X} x \cdot s(x,y)}{\sum_{Y}\sum_{X} s(x,y)} = \frac{m_{10}}{m_{00}} \qquad x_t = \frac{\sum_{Y}\sum_{X} x \cdot s(x,y)}{\sum_{Y}\sum_{X} s(x,y)} = \frac{m_{01}}{m_{00}} \qquad (13)$$

Centrální moment řádu $p+q$:

$$\mu_{pq} = \sum_{Y}\sum_{X} (x - x_t)^p \cdot (y - y_t)^q \cdot s(x,y) \qquad (14)$$

Orientace objektu:

$$\varphi = 0.5 \cdot \arctan\left(\frac{2\frac{\mu_{11}}{m_{00}}}{\frac{\mu_{20}}{m_{00}} - \frac{\mu_{02}}{m_{00}}} \right) \qquad (15)$$

Radiometrické deskriptory založené na regionech bylo nutno implementovat. Pro výpočet prostorových a centrálních momentů je možno využít funkce poskytované knihovnou OpenCV.

Použití deskriptorů a momentů je pouze základní. Momentů by bylo možno využít pro přesnou identifikaci objektů. Jejich plnému využití však brání fakt, že většina objektů ve scéně má podobný tvar.

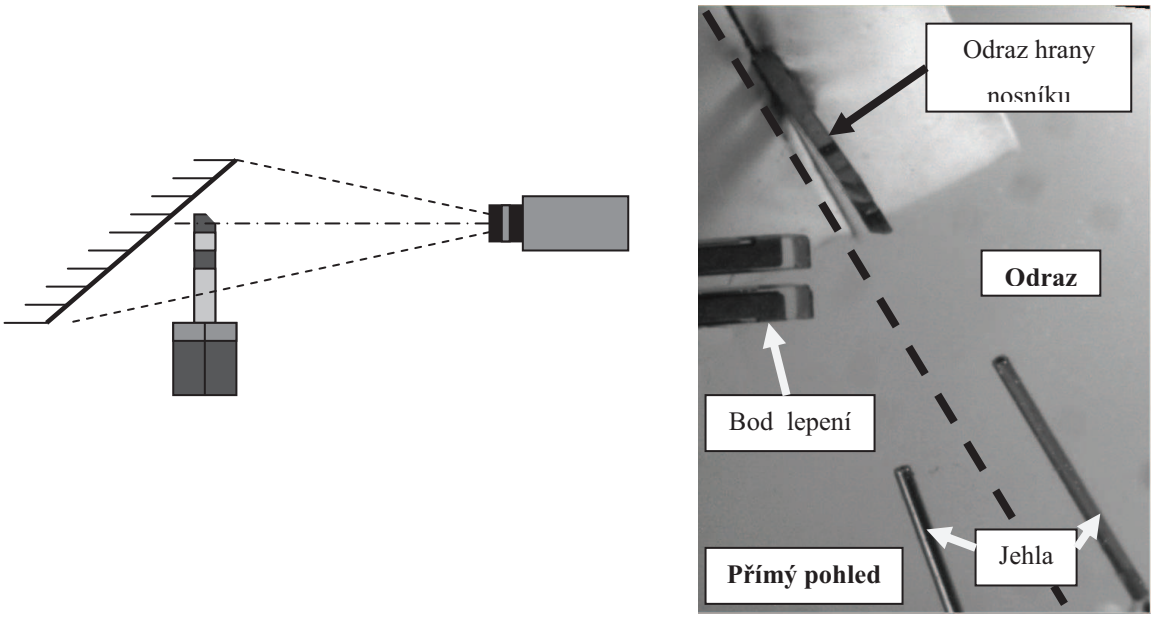
V projektu není použit žádný klasifikátor. Využití deskriptorů závisí na různých potřebách jednotlivých zpracování a jeho logika většinou vychází z uspořádání scény a předpokládaných vlastností hledaných objektů. Dá se říci, že je použito rozhodovací pravidlo specifické pro každou část úlohy.

4.3 PROBLEMATIKA LEPENÍ HROTU

Na Obr. 11 je naznačen způsob snímaní scény odrazem od zrcadla pro zajištění informace o hloubce. Scéna je skutečně natočena o 90° jak je naznačeno na obrázku, pro zpracování je však toto natočení kompenzováno otočením kamery.

Nanesení lepidla a montáž hrotu jsou prakticky totožné operace (Obr. 12, Obr. 13). Při nanášení lepidla je ploška, na níž se má provést montáž, obrácena směrem dolů (Obr. 11) a lepidlo je nanášeno na vodivou plochu dutou jehlou. Při lepení drátku je ploška s naneseným lepidlem naváděna tak, aby byla pomocí lepidla spojena s drátkem – budoucím hrotem. Po navedení nosníku do správné pozice vůči drátku je lepidlo vytvrzeno horkým vzduchem.

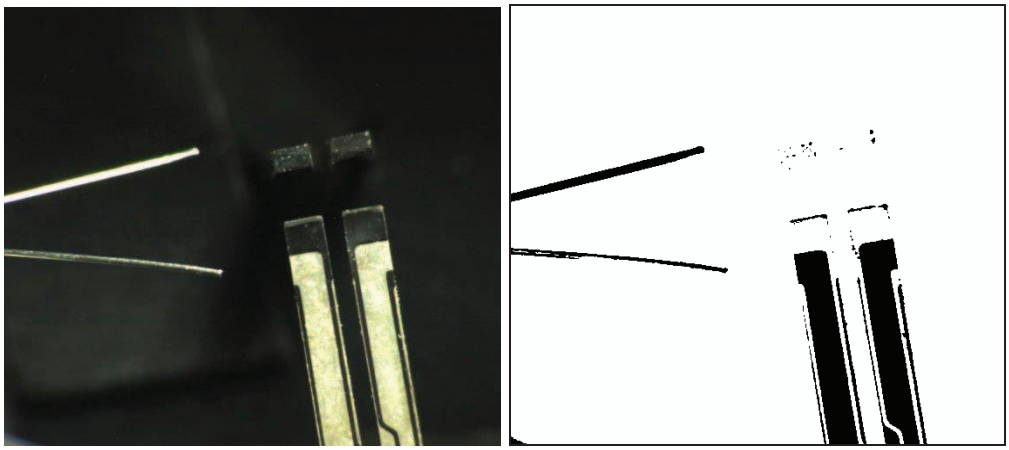
Na Obr. 12 je scéna zachycující úlohu lepení drátku pořízená v laboratorních podmínkách za původního předpokladu černého pozadí. Scéna je snímána jednou kamerou a pomocí zrcátka je zobrazen vrchol nosníku, respektive jeho hrana v reálných podmínkách.(Obr. 11, Obr. 13)



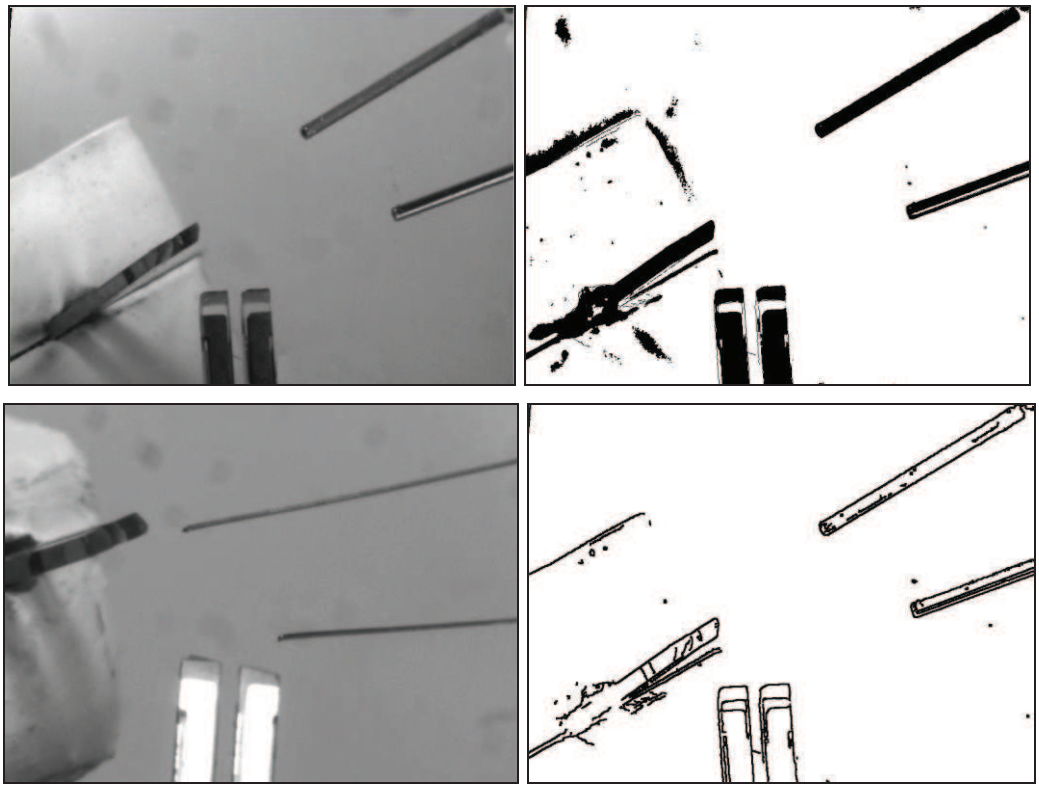
Obr. 11: Schéma způsobu snímání obrazu scény, schematický pohled zhora a pohled respektující skutečnou orientaci scény

Ze snímku je patrné, že dobrou volbou pro segmentaci drátků nebo jehly bude prahování. Vrchol nosníku snímaný odrazem od zrcadla (Obr. 12) není možno vzhledem ke špatnému kontrastu prahovat. Ve skutečnosti však není kamera umístěna přesně proti zrcadlu, ale svírá s ním, z důvodu konstrukčního řešení stroje, určitý úhel. Díky tomu je možno pro detekci osy vrcholu využít boku nosníku. Rozdíl mezi laboratorními podmínkami a skutečnou snímanou scénou je patrný z porovnání obrázků Obr. 12 a Obr. 13. V laboratorních podmínkách by bylo možno

pro segmentaci volit jeden práh, například procentuální vycházející z plochy objektů ve scéně, pro celý obraz. V reálných podmínkách bylo použito adaptivní prahování z důvodu nerovnoměrného nasvětlení scény.



Obr. 12: Snímaná scéna v laboratorních podmínkách a její prahovaný obraz



Obr. 13: Reálná scéna s jehlou, adaptivně prahovaný obraz, montáž drátku a obraz hran získaný Cannyho hranovým detektorem

4.4 DRÁTEK A JEHLA

Jak již bylo uvedeno výše, bylo použito popisu spíše pro ověření správnosti nalezeného objektu. Při hledání objektu drátku, také jehly, bylo využito znalosti objektu a rozložení scény. Vychází se při tom z obrazu vzniklého adaptivním prahováním, nebo sloupcovým adaptivním prahováním (adaptivní prahování provedené na základě průměrného jasu sloupce, nikoliv čtvercového okolí). Objekt, který může být drátkem nebo jehlou, musí splnit následující podmínky:

- Musí se nalézat v pravé části obrazu.
- Musí být podlouhlý.
- Objekt musí svírat se spodní hranou obrazu úhel přibližně 22°. (vychází z konstrukce a vzájemné polohy kamery a scény, může se však měnit v závislosti na poloze kamery)

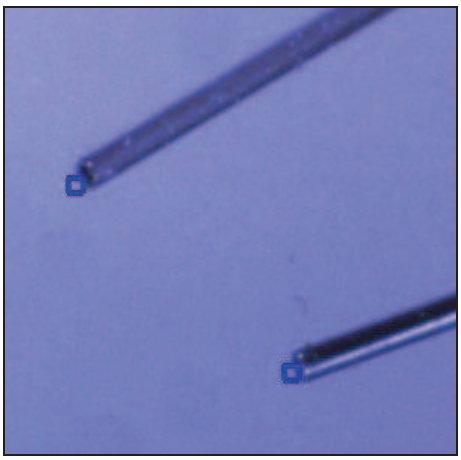
Pro testování podmínek je využita orientace zjištěné z momentů, podlouhlosti a jsou hledány pouze objekty v pravé straně obrazu – nalezením všech barev ve vymezené oblasti. Všechny parametry a rozsahy tolerancí popisu jsou v software nastavitelné pomocí konfiguračních souborů nebo pomocí uživatelského konfiguračního rozhraní.

Pokud je nalezeno méně odpovídajících objektů než 2 (objekt a jeho odraz v zrcadle), pak došlo k chybě. Může chybět drátek nebo nedošlo k dojezdu na správnou pozici. V případě většího počtu došlo k výskytu cizího objektu ve scéně nebo nesprávné segmentaci. Vzhledem k malým rozměrům objektů může být takovým objektem i prach nebo vlas. Program vybere dva nejpravděpodobnější objekty, na základě statistických vlastností jako je délka, průměrná tloušťka a její rozptyl a dalších, a upozorní obsluhu.

Nejdůležitější informace je poloha konce drátku, který má být přilepen k nosníku. Objekt je v tomto případě tenký a dlouhý, nejjednodušším způsobem nalezení konce drátku je proložení polynomem prvního nebo druhého stupně. Bod na proložené křivce, kde dochází k přechodu mezi objektem a pozadím, je koncem drátku. Konkrétně v tomto případě je použito proložení přímkou v poslední třetině délky drátku a nalezení koncové hrany objektu. Proložení přímkou lze provést pomocí lineární regrese.

$$y = ax + b \tag{16}$$

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2} \qquad b = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2} \tag{17}$$



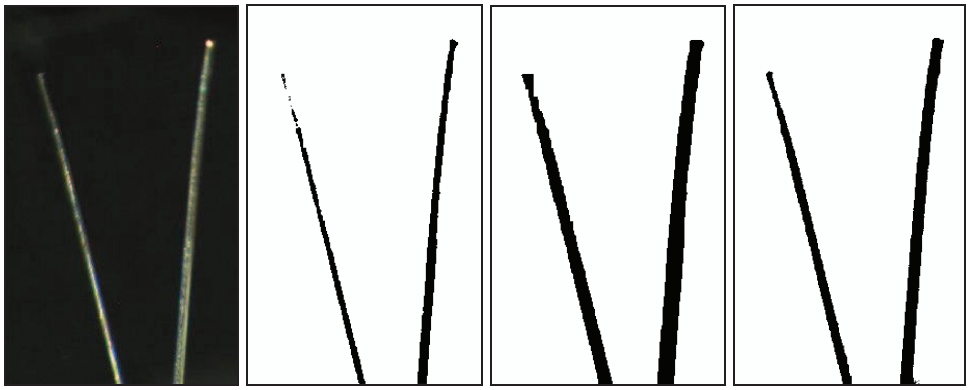
Obr. 14: Poloha konečných bodů jehly

Problémem při segmentaci může být nehomogenita drátku nebo jeho špatné nasvícení. Při prahování jsou pak získány dva nebo více objektů - drátek je přerušený. Tím by samozřejmě došlo k detekci konce drátku dříve a k nesprávnému vyhodnocení polohy jeho konce. Pro spojení takovýchto objektů je možné proložit objekt polynomem 2. stupně a v okolí výsledné křivky vyhledávat místa, která by mohla být pokračováním drátku. Příslušnost k drátku je možno testovat podle jasu v šedotónovém obrazu, nebo spojovat objekty v binárním obrazu.

Jedním z možných přístupů je také druhé prahování s dvojím prahem v okolí této křivky. První práh je globální, druhý práh je menší. Bod je pak uznán jako objekt, pokud je jeho jas vyšší než jas nižšího prahu a zároveň se v jeho okolí vyskytuje bod, který již byl uznán jako bod náležející objektu. Další možností je provést v okolí křivky lokální prahování.

Oba předchozí přístupy předpokládají, že drátek je již nalezen a proložen přímkou. V již segmentovaném obrazu lze použít, ještě před roztříděním na objekty, matematickou morfologii. Dilatace následovaná vhodnou erozí dokáže spojit drobná přerušení drátku v prahovaném obrazu. Na Obr. 15 byl použit strukturní element velikosti 14x14 při dilataci a následně sloupcový element délky 14 pro erozi, aby

byla zachována délka drátku. Velikost elementu je volena na základě známého měřítka obrazu – 14 bodů odpovídá 140μm, drátek má tloušťku 100μm. Při dilataci je však nutné dávat pozor, aby nedošlo ke spojení nesprávných objektů.



Obr. 15: Špatně nasvícený drátek, nesprávná segmentace, použití matematické morfologie, použití dvojího prahu (okolí 5x5)

4.4.1 Alternativní segmentace pro nalezení drátku

Drátek nebo jehla jsou objekty, jejichž zpracování je založeno na segmentaci prahováním. K zvýraznění drátku pro jeho segmentaci lze přistupovat i pomocí gradientních metod a konvoluce. Lze použít sloupcovou masku, jejíž velikost respektuje šířku drátku a jejíž součet členů je roven nule. Takovou maskou může být například vektor, jehož členy tvoří obdélníkovou funkci s nulovou střední hodnotu, kde šířka obdélníkové části je podobná šířce drátku. Jak je uvedeno v následující kapitole, tento postup nevykazoval výraznější zlepšení oproti použití adaptivního prahování a nebyl při řešení využit.

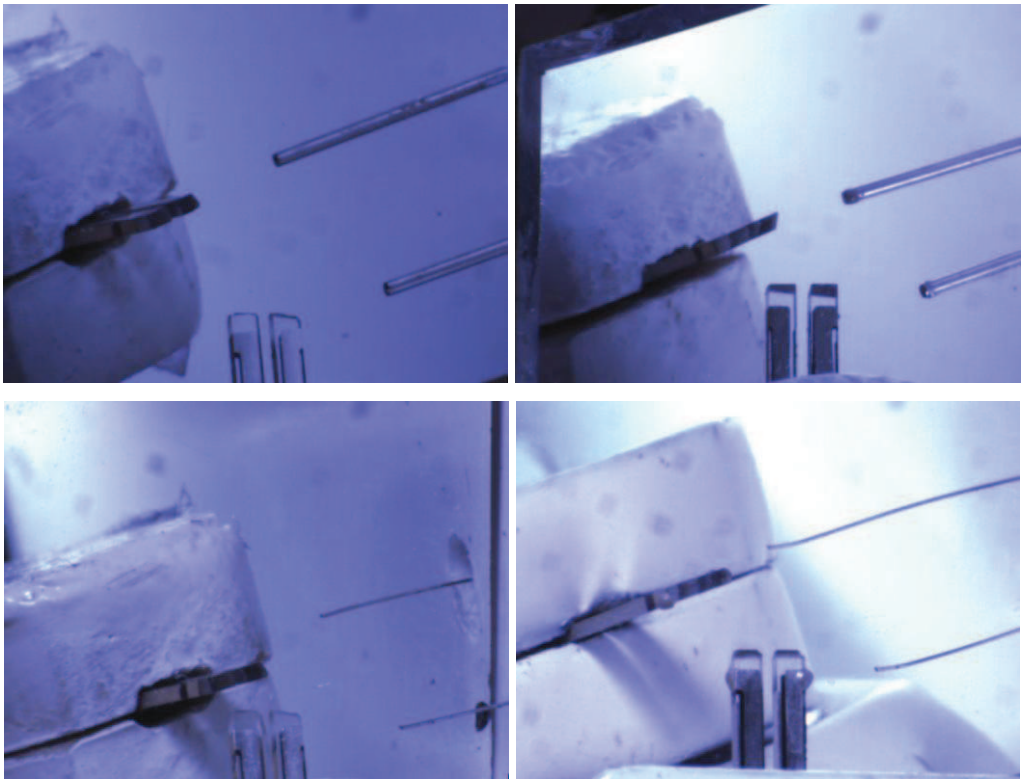
4.4.2 Drátek a jehla – shrnutí

Při zkouškách byly testovány tři segmentační metody, nebyl použit žádný princip spojování potenciálně přerušeného drátku:

- Adaptivní prahování.
- Adaptivní prahování ve sloupcích.
- Konvoluce s následným prahováním.

V Tabulka 1 je úspěšnost jednotlivých metod v testech na 110 testovacích snímcích získaných během vývoje projektu. Procentuální úspěšnost vyjadřuje spíše

robustnost ke změně podmínek (viz Obr. 16), nežli přímo spolehlivost algoritmů. Testovací snímky se liší v poloze objektů, jejich orientaci, nastavení zrcadel, nastavení světel, stínech, viditelných cizích objektech, přesnosti uchycení nosníku v magnetickém držáku a dalších parametrech. Ve finálním uspořádání jsou tyto parametry stálé a úspěšnost by se měla při správném nastavení blížit 100%.



Obr. 16: Příklady testovacích snímků

Způsob zpracování	Úspěšnost původní	Úspěšnost zlepšená
Adaptivní	90%	95%
Konvoluce	89%	94%
Adaptivní sloupcově	90%	95%

Tabulka 1: Úspěšnost při vyhledávání polohy konce drátku nebo jehly

Prakticky existují dvě hlavní chyby při hledání konce drátku nebo jehly. První chybou je spojení hledaného objektu s jiným objektem v obrazu. Tato chyba vzniká stínem, nečistotu nebo nesprávnou délkou drátku a není ji možno úplně eliminovat. Druhou hlavní chybou je záměna objektů, například s hranou zrcadla viditelnou ve

scéně. Tato chyba byla eliminována změnou rozložení scény a zvětšením zrcadla. Úspěšnosti po provedených úpravách odpovídá poslední sloupec Tabulka 1.

Nejlepší metodou je sloupcové adaptivní prahování, protože není citlivé na svislé jasové přechody vznikající u hran zrcadla a vykazuje stejnou spolehlivost jako blokové adaptivní prahování.

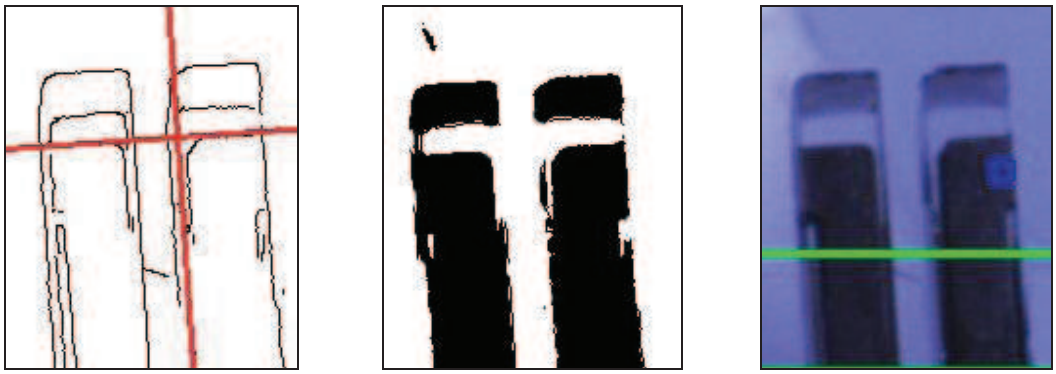
4.5 NOSNÍK A NALEZENÍ BODU PRO LEPENÍ

4.5.1 Nosník a bod pro lepení pomocí prahování

Nosník je reprezentován hlavně vodivými cestami na jeho povrchu. Dvě hlavní vodivé cesty tvoří objekty s největší plochou ve spodní části snímku. Přibližná poloha je určena oblastí zájmu v dolní části obrazu. V rámci této oblasti jsou nalezeny všechny objekty, získané segmentací adaptivním prahováním (Obr. 13), které se jí alespoň dotýkají. S výhodou je využito podobnosti obou konců nosníku a faktu že tvoří dvojici. Pro rozeznání obou vodivých cest jsou využity následující podmínky:

- Oba objekty mají podobnou plochu.
- Objekty (jejich těžiště) jsou vzdáleny 9μm.
- Objekty jsou kolmé na spodní hranu obrazu.
- Objekty mají podobnou podlouhlost a stejný tvar.

Informací, kterou je nutno získat, je poloha vodivé plošky na pravé straně nosníku. Pomocí Houghovy transformace je možno získat přímku vymezující horní a levou hranu vodivé cesty. Určení bodu pro lepení je dáno posunem směrem dolů od horní přímky a vpravo od levé přímky. Tento bod odpovídá středu plochy, na kterou má být provedena montáž. Vychází se ze znalosti rozměrů nosníku a znalosti převodní konstanty. Přesná poloha je nastavitelná podle požadavků obsluhy. Výsledek tohoto postupu je na Obr. 17.



Obr. 17: Hrany nosníku a okrajové přímky, adaptivně prahovaný obraz a označená část nosníku pro montáž hrotu

4.5.2 Nosník a bod pro lepení pomocí korelace

Metoda prezentovaná v předchozí kapitole vykazuje dobré výsledky pouze za předpokladu dokonalé segmentace obrazu prahováním. Během praktických testů docházelo často k nedokonalé segmentaci. Důvodem chyby v segmentaci byly buď cizí objekty jako prach a nečistoty na zrcadle, nebo nedokonalé nasvětlení scény. Protože pro aplikaci je podstatná vysoká robustnost a nelze zajistit bezprašné prostředí, stejně jako nelze eliminovat neodborný zásah do osvětlení nebo okolních světelných podmínek, bylo nutno algoritmus přepracovat.

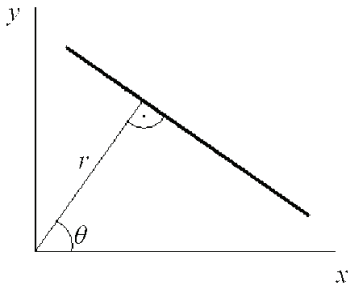
Obraz hran získaný Cannyho hranovým detektorem je poměrně kvalitní a dobře zachycuje hranice nosníku i v případě změny osvětlení nebo výskytu menších cizích objektů. Možnou metodou použitelnou k nalezení nosníku je korelace. Přímé použití korelace k hledání nosníku v obrazu hran nebo prahovaném obrazu však není možné z několika důvodů:

- Vysoká závislost na orientaci objektu, ještě více umocněná použitím obrazu hran.
- Vzhledem k přesnosti uchycení různá délka a orientace nosníku v obrazu.
- Problematické získávaní vzoru pro korelaci z důvodu možnosti použití různého zvětšení a různé velikosti kalibračního vzoru vedoucí k proměnné velikosti nosníku v obrazu.
- Nedostatečná přesnost korelace při použití prahovaného obrazu.

Tolerance orientace nosníku je vzhledem ke způsobu uchycení přibližně v rozsahu ±5°, délka nosníku ±20%. Přímé použití korelace je proto problematické. Je však možno použít korelaci založenou na výstupu Houghovy transformace, je ale nutné přepracování takto získaných dat.

Výstupem Houghovy transformace jsou přímky odpovídající hranám nosníku a vodivých cest. Pokud je práh pro výběr přímek z prostoru parametrů nastaven nízko, je získán velký počet přímek. Existuje tak dostatečná rezerva i v případě změny parametrů obrazu. Ze získaných přímek je nutno určit nejpravděpodobnější polohu nosníku. Velice snadno lze vybrat všechny přímky, jejichž směrnice odpovídá předpokládané orientaci nosníku. V Houghově transformaci je přímka reprezentována úhlem θ mezi osou x a kolmicí k přímce a délkou této kolmice r do počátku souřadného systému. ([4], str. 114)

$$r = x \cdot \cos\theta + y \cdot \sin\theta \qquad (18)$$



Obr. 18: Reprezentace přímky pro Houghovu transformaci [4]

Histogram lze vytvořit ze vzdáleností r jednotlivých přímek vybraných podle předpokládaného úhlu θ hran nosníku a měl by obsahovat čtyři maxima odpovídající polohám jeho jednotlivých hran (Graf 3). Histogram je možno vytvořit za předpokladu malé tolerance akceptovaného úhlu θ přímek. Při tvoření histogramu totiž dochází ke ztrátě informace o tomto úhlu.

Model nosníku lze pro korelaci vytvořit velice přesně. Vzájemné vzdálenosti maxim odpovídají fyzickým rozměrům nosníku, jež jsou známé, stejně tak jsou známé rozměry obrazu. Model odpovídající parametrům nosníku je již možné korelovat s histogramem délek kolmic přímek. Normovaná křížová korelace v prostorové rovině je výpočetně náročná, rychlejší je výpočet ve frekvenční rovině a

zpětná transformace do prostorové roviny.([8]) Obecně pro dvourozměrnou korelaci:

$$C(x,y) = F(x,y) \cdot G^{*}(x,y) \qquad (19)$$

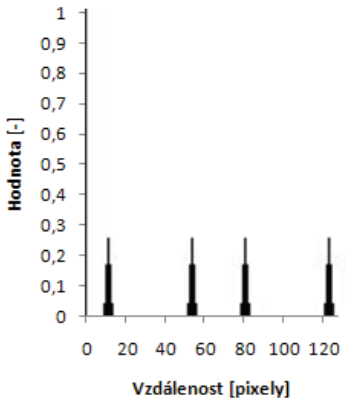
$C(x,y)$ - frekvenční spektrum výsledné korelace.

$F(x,y)$ - frekvenční spektrum obrazu.

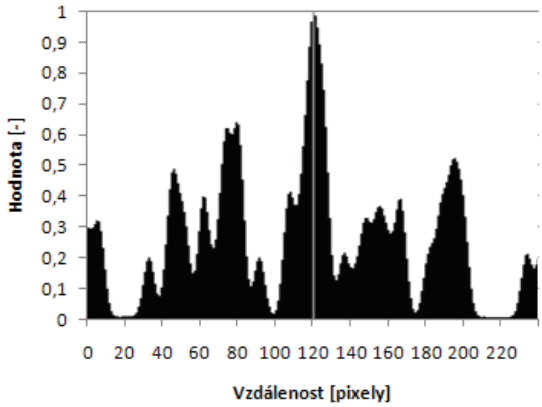
$G^{*}(x,y)$ - komplexně sdružené frekvenční spektrum vzoru.

Celé zpracování ilustrují grafy níže. Model respektuje možné nepřesnosti kalibrace nebo výrobní tolerance nosníku. Poloha nosníku je určena místem největší shody modelu a histogramu přímek.

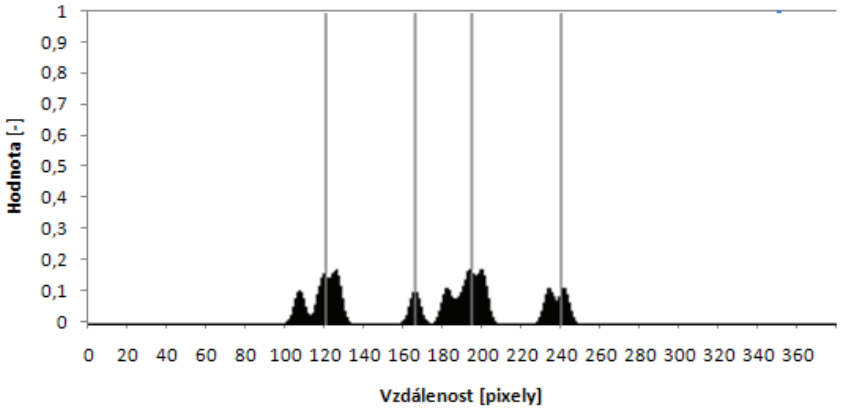
Graf 1: Model nosníku



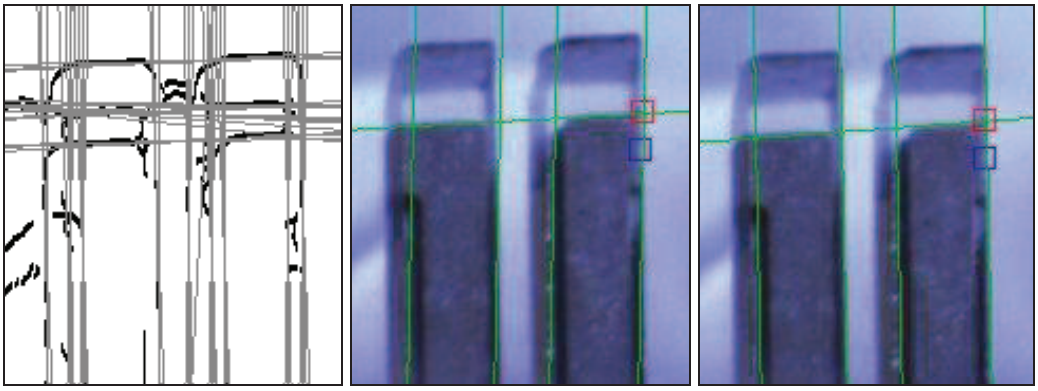
Graf 2: Výsledek korelace



Graf 3: Normalizovaný histogram počtu přímek v závislosti na *r*, vyznačena nejlepší shoda s modelem



Finálním krokem je nalezení nejnižší přímky kolmé na hrany nosníku. Tato přímka reprezentuje horní hranu vodivé cesty. Nalezení bodu lepení je provedeno změřením vzdálenosti od průsečíku horní přímky a poslední hrany. Výsledek zpracování je na Obr. 19, horní značka reprezentuje průsečík okrajových přímek, dolní bod pro lepení.



Obr. 19: Všechny možné hraniční přímky, místo pro montáž hrotu nalezené pomocí korelace a pomocí měření vzdálenosti

Metoda korelace vykazuje dobrou robustnost. Přesnost je dána hlavně rozptylem přímek získaných pomocí Houghovy transformace, nicméně pro zpracování je dostatečná. Přesnou polohu hranice lze korigovat dohledáním hrany nosníku.

4.5.3 Nosník a bod pro lepení pomocí měření vzájemné vzdálenosti

Během testů se při korelaci jako problematická ukázala situace, kdy je některá z hran nosníku, ať z důvodu nasvícení nebo defektu nosníku, výrazně silnější, než hrany ostatní, nebo několikanásobná. V takovém případě může dojít k nesprávné korelaci a posunu předpokládané polohy o přibližné násobky ¼ modelu. Taková chyba by byla při výrobě kritická.

Místo korelace lze použít měření vzdálenosti okrajových přímek nosníku od jejího modelu. Pro každou hypotézu polohy nosníku je měřena vzdálenost modelu, tvořeného vektorem relativních poloh jeho hranic, od přímek v obrazu. Hypotéza

s nejmenší hodnotou vzdálenosti odpovídá, s největší pravděpodobností, správné poloze nosníku. Výpočet vzdálenosti popisuje rovnice:

$$H = \min_{i=1..a}\left(\sqrt{(m_1 - h_{1,i})^2} + \ldots + \sqrt{(m_n - h_{n,i})^2}\right) \qquad (20)$$

n – rozměr modelu

a – počet možných hypotéz polohy nosníku

m,h – parametr r jednotlivých přímek modelu a hypotézy

Výhodou postupu je možnost kvantifikovat míru shody a případná možnost odmítnutí správnosti nalezené polohy na základě tohoto výsledku. Výstupem korelace je normované maximum, jehož hodnota je rovna 1. Určení správnosti změřené polohy by vyžadovalo složitější zpracování. Měření vzdálenosti je v tomto případě výhodnější. Zjištěný sklon hraničních přímek je přesnější než u korelace, kde o tuto informaci přicházíme při vytváření histogramu. Při měření vzdálenosti je vybrána čtveřice přímek nejpodobnější modelu, tyto přímky respektují hrany objektu. Při korelaci je určena poloha hran pro nejlepší shodu s modelem. Model nemusí přesně respektovat hrany objektu. Proto je určení hran nosníku nalezené měřením vzdálenosti od modelu přesnější. Rozdíl je patrný na Obr. 19.

Nevýhodou měření vzdálenosti je velká výpočetní náročnost algoritmu. Při „hrubém“ přístupu je nutné provést $\binom{n}{a}$ jednotlivých výpočtů vzdálenosti, kde n je počet nalezených přímek a a je rozměr modelu. Toto množství lze snížit s ohledem na apriorní znalosti o rozložení scény. Počet nutných výpočtů je však stále vysoký a stoupá s počtem nalezených přímek a složitostí modelu. V tomto ohledu je lepším přístupem korelace, protože probíhá pouze mezi dvěma vektory a je velice rychlá.

4.5.4 Nosník a bod pro lepení – shrnutí

Pro rozhodnutí o tom, který algoritmus použít, je nejdůležitějším kritériem robustnost, teprve na druhém místě je rychlost zpracování obrazu. K určení nejlepšího postupu byl proveden srovnávací test s cílem vyhodnotit úspěšnost

zpracování a rychlost. Ve všech obrazech se hledaný objekt vždy nacházel, ale podmínky snímání byly různé. Nastavení algoritmů bylo neměnné pro všechny provedené testy. Za úspěšné nalezení byl považován stav, kdy nedošlo k významnému odchýlení od správného bodu lepení. Výsledky testu jsou v Tabulka 2. Testovací obrazy jsou součástí elektronické přílohy práce.

Způsob zpracování	Úspěšnost	Chyba okraj	Chyba vrchol	Průměrný čas
Korelace	89%	3%	8%	23 ms
Vzdálenost	91%	2%	7%	31-300 ms

Tabulka 2: Úspěšnost při vyhledávání polohy montážního bodu

Procentuální úspěšnost opět vyjadřuje spíše robustnost ke změně podmínek. Nejčastější chybou je nenalezení vrcholové hrany vodivých cest. Příčinou je špatný kontrast v obrazu z důvodu stínu nebo špatného nasvícení v oblasti nosníku. Při osvětlování scény je nutné eliminovat tento vliv. Vrcholová hrana je také kratší, nežli okrajové hrany, což znesnadňuje nalezení přímky ve výstupním prostoru parametrů Houghovy transformace. Měření vzdálenosti vykazuje nižší náchylnost k nesprávnému nalezení okrajů nosníku a jejich poloha je přesnější. Časová náročnost se však velmi různí v závislosti na počtu přímek vybraných z výstupu Houghovy transformace. Prahování je velmi nespolehlivé a nebylo testováno. Odchylka nalezeného montážního bodu od jeho správné polohy se pohybuje, pro zpracování korelací i měřením vzdálenosti, v jednotkách bodů (odpovídá desítkám μm).

4.6 VRCHOL NOSNÍKU

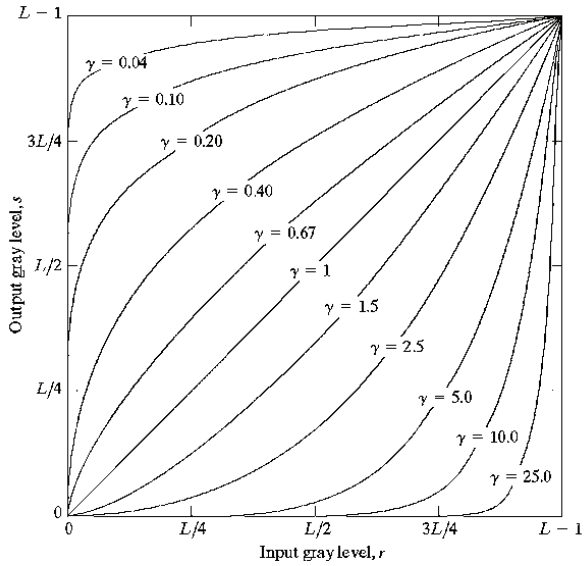
Při zpracování vrcholu nosníku je postačující informací popis jeho osy. Vzdálenost mezi drátkem a vodivou plochou je v osách x a y určena z předchozího zpracování. Poslední chybějící informací je vzdálenost v hloubce zachycená odrazem v zrcadle.

V laboratorních podmínkách bylo velkým problémem vrchol nosníku v obrazu najít z důvodu špatného kontrastu. V úvahu přicházely jasové korekce a využití přibližné informace o jeho poloze v ose nosníku. Možné použití nelineární

jasové korekce je na Obr. 20 a Obr. 21: Výsledek aplikace korekce. V tomto případě byla zvolena korekce ve tvaru ([2] str. 80) :

$$s = cr^{\gamma} \qquad (21)$$

Kde s je bod výstupního obrazu a r je bod vstupního obrazu. Tato korekce, s hodnotu koeficientu γ větší než jedna, potlačí tmavé pozadí a zvýrazní o něco světlejší vrchol nosníku. Z matematického hlediska však takováto transformace nepřináší informační zisk. Právě naopak, dochází ke ztrátě informací transformací jasové stupnice. Volba koeficientů rovněž závisí na momentálním nastavení osvětlení a clony kamery, což není z hlediska zpracování optimální.



Obr. 20: Křivky nelineární korekce jasu [2]

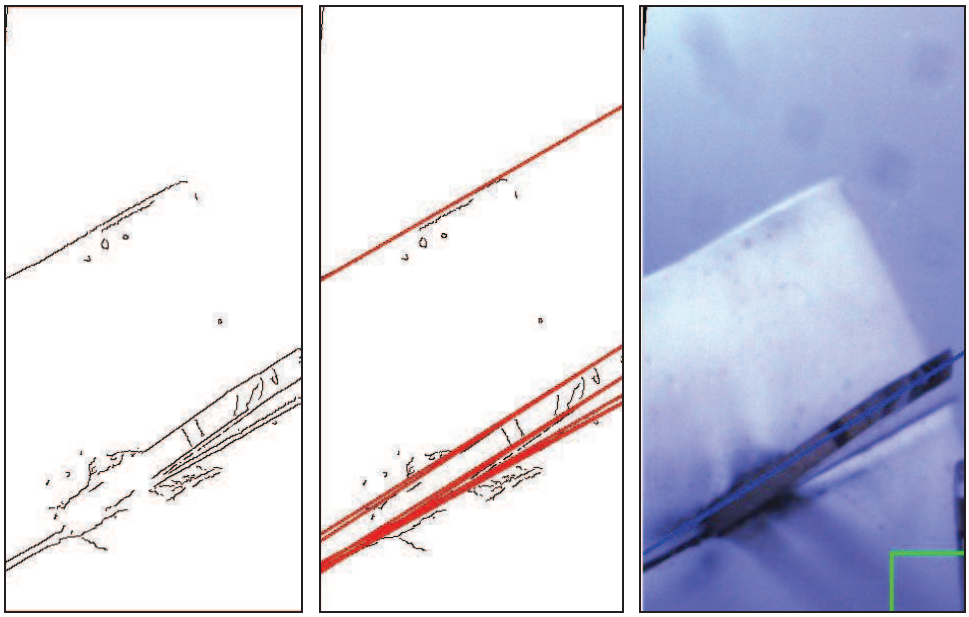
Obr. 21: Výsledek aplikace korekce

Situace se značně zjednodušila během implementace algoritmů na zařízení. Jak je patrné z Obr. 13, je z konstrukčního důvodu uspořádání scény jiné. Důvodem je uchycení nosníku pod úhlem 22° k vodorovné ose zrcadla. Místo vrcholu je

v zrcadle viditelný bok nosníku s kontrastní vodivou cestou. Opět platí, že stačí zjistit polohu podélné osy objektu. Existují dvě metody získání osy objektu implementované v programu s možností volit, která bude použita.

První metoda vychází z nalezení objektu a poté jeho proložení přímkou pomocí lineární regrese. Tento postup předpokládá dokonalou segmentaci obrazu. Jakákoliv chyba ve tvaru oblastí vzniklé při segmentaci pozmění orientaci osy. Metoda proto není vhodná, pokud není možno zajistit kontrast ve scéně správným osvětlením nebo dokonalou segmentací.

Druhou metodou je využití obrazu hran. Hrany získané aplikací Cannyho hranového detektoru je možné zpracovat Houghovou transformací. Při výběru přímek z prostoru parametrů je s výhodou možno využít znalosti přibližné orientace boku nosníku v obrazu. I přesto je výsledkem mnoho přímek, opět je cílem zachovat rezervu jako ochranu před změnou parametrů snímané scény, z nichž ne všechny náleží hledanému objektu. Další informací, kterou je nutno využít pro identifikaci odpovídajících hran, je jejich vzdálenost - tloušťka nosníku je 350μm. Všechny dvojice, které projdou výběrem, jsou potenciálními okraji objektu. K výskytu více dvojic může dojít, pokud je nosník mírně nakloněn v obrazu a jsou viditelné další hrany objektu, nebo pokud mají výběrové parametry nastavený vyšší akceptovaný rozsah. Výběr nejpravděpodobnějšího směru osy je možno provést vážením jednotlivých dvojic přímek počtem bodů hrany, kterou protínají. Je nutno podotknout, že „výběrem“ je myšlena korekce směru a polohy v řádu maximálně jednotek stupňů a pixelů. Tato metoda je výpočetně náročnější, avšak podstatně spolehlivější než pouhé proložení přímkou v případě prahovaného obrazu.



Obr. 22: Výsledek zpracování vrcholu pomocí Houghovy transformace

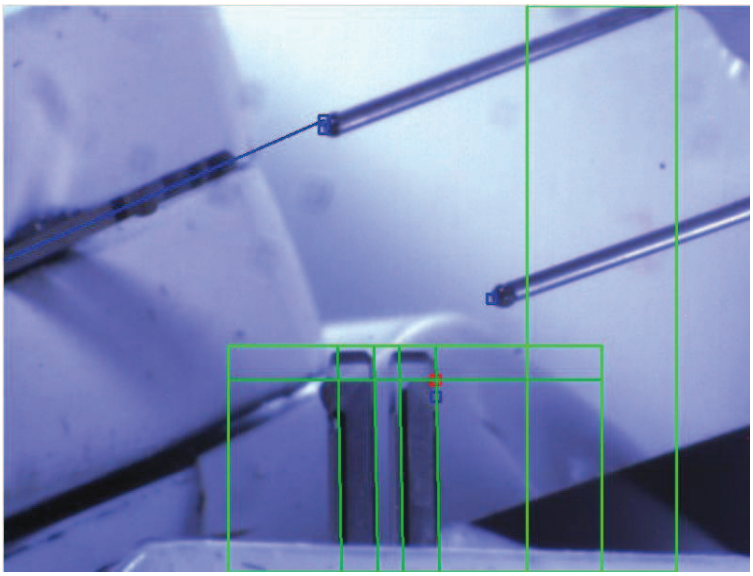
Tabulka 3 porovnává spolehlivost obou přístupů za různých světelných podmínek a rozložení scény. Zpracování pomocí hran může selhat pouze v případě výskytu objektu o šířce srovnatelné s šířkou nosníku nebo při vysoké míře šumu v obrazu. Většina šumu je však odstraněna při předzpracování.

Způsob zpracování	Úspěšnost
Prahování a lineární regrese	55%
Hrany a HT	98%

Tabulka 3: Úspěšnost při hledání osy nosníku

4.7 SHRNUTÍ ZPRACOVÁNÍ PŘI LEPENÍ HROTU

Výsledek popsaných postupů je na Obr. 23. Ze známých pozic jednotlivých dílů je možno určit vzdálenosti převodem pixel-délka. Tuto vzdálenost v osách *x*,*y*,*z* lze použít pro řízení krokových motorů v jednotlivých osách.



Obr. 23: Výsledek zpracování scény lepení drátku / nanášení lepidla spolu s označnými oblastmi vyhledávání nosníku a drátku

Postup zpracování dobře funguje na statickou scénu. Protože zpracování obrazu zabere poměrně dlouhý čas (Tabulka 4), není možno jej použít jako přímou zpětnou vazbu pro řízení. Během montáže je při přibližování postupováno v několika krocích. Každému kroku předchází zpracování obrazu a výpočet vzdáleností objektů. Pohyb v ose *z* (hloubka) je prováděn vždy samostatně a vícekrát. Protože se drátek před zpracováním nenachází v ose nosníku, není pro něj plně platná převodní konstanta získaná při kalibraci obrazu. Převodní konstanta a korekce je vypočtena v ose nosníku. Jejím použitím pro předmět nacházející se mimo rovinu nosníku proto dochází k chybě měření prespektivou. Tato chyba je však při malých vzdálenostech od roviny scény zanedbatelná a zmenšena přiblížením v několika krocích.

V Tabulka 4 je uvedena časová náročnost jednotlivých částí zpracování obrazu. Časy vybraných algoritmů jsou uvedeny **tučně.** Sloupcová segmentace adaptivním prahem je výrazně pomalejší, protože v OpenCV existuje pouze bloková implementace adaptivního prahování. Obraz bylo nutno rozdělit do sloupců a provést adaptivní prahování pro každý sloupec zvlášť. Toto rozdělení podstatně prodloužilo čas nutný pro zpracování obrazu při segmentaci.

Krok zpracování	Čas [ms]	Zahrnuje
Kalibrace	**196**	Prahování, vpočet korekcí, transformace obrazu
Předzpracování	**101**	Šedotónová konverze, filtrace mediánem, Cannyho hranový detektor
Segmentace (sloupcově)	**664**	Sloupcové adaptivní prahování, barvení, oddělení a popis objektů
Segmentace (blokově)	*211*	*Blokové adaptivní prahování, barvení, oddělení a popis objektů*
Drátek/jehla	**24**	Dodatečný popis, nalezení nejpodobnějších objektů, proložení, nalezení okrajové hrany
Nosník (vzdálenost)	**31 – 300**	Houghova transformace, nalezení přímek, výpočet vzdáleností, nalezení vrcholové přímky
Nosník (korelace)	*23*	*Houghova transformace, nalezení přímek, vytvoření histogramu a korelace, nalezení vrcholové přímky*
Vrchol	**43**	Houghova transformace, dvojice, vážení přímek
celkem	**1057 – 1357**	Pro sloupcové prahování a hledání nosníku pomocí měření vzdálenosti

Tabulka 4: Časy potřebné pro zpracování

4.8 LEPTÁNÍ

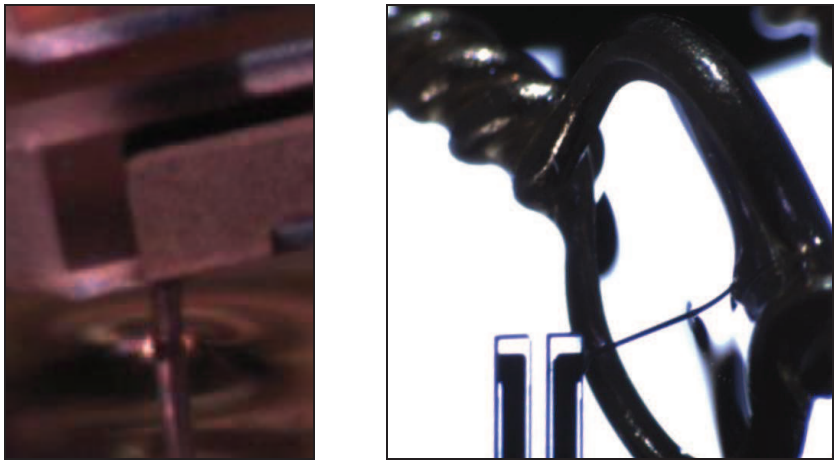
Po přilepení drátku je dalším krokem jeho leptání, aby došlo k vytvoření tenkého hrotu. Tento hrot by měl být co nejkratší, aby společně s lepidlem co nejméně narušoval rovnováhu nosníku. Zároveň však při ponořování hrotu do leptacího roztoku nesmí dojít ke kontaktu roztoku s lepidlem nebo nosníkem. Lepidlo obsahuje částečky stříbra pro zajištění elektrické vodivosti. Pokud dojde ke kontaktu s leptacím roztokem, tak se tyto částečky uvolní a při leptání způsobí špatné, nerovnoměrné odleptání hrotu a výrazně sníží kvalitu výsledného výrobku.

Scéna snímaná při leptání je složitější než v předchozích krocích. Obsahuje množství odlesků a hranice kapaliny není jasně patrná.(Obr. 24) Dochází také k různému chování hladiny podle směru pohybu drátku. Při zasouvání se hladina prohýbá směrem dolů, při vytahování vlivem vzlínavosti směrem vzhůru. Hloubku ponoření je nutno měřit mezi hranou nosníku a povrchem kapaliny. Cílem je tedy přesně zjistit místo průniku drátku skrze hladinu leptacího roztoku.

Při zpracování bylo použito nasvícení odzadu pomocí odrazu od bílého pozadí. Poloha okraje nosníku se, díky fyzickému svázání s kamerou, nemění a je

tedy známa z předchozího zpracování. Pro zjištění vzdálenosti hladiny je využito jevu, kdy dochází, vlivem povrchového napětí, k deformaci hladiny v okolí drátku.(Obr. 24) Tato deformace odráží světlo směrem od kamery a je určitelná poklesem jasu v okolí drátku. Drátek je nalezen pomocí Houghovy transformace v oblasti hladiny s využitím znalosti o jeho předpokládaném směru. Celý postup v krocích:

1. Nalezení oblasti hladiny a vytvoření její masky.
2. Nalezení drátku ve vymezené oblasti.
3. Nalezení oblasti poklesu průměrné hodnoty jasu odpovídající kontaktu drátku s hladinou.
4. Výpočet odpovídající vzdálenosti mezi okrajem nosníku a hladinou a navedení nosníku do vzdálenosti požadované délky hrotu.



Obr. 24: Deformace hladiny v oblasti průniku drátku hladinou leptacího roztoku a snímaná scéna

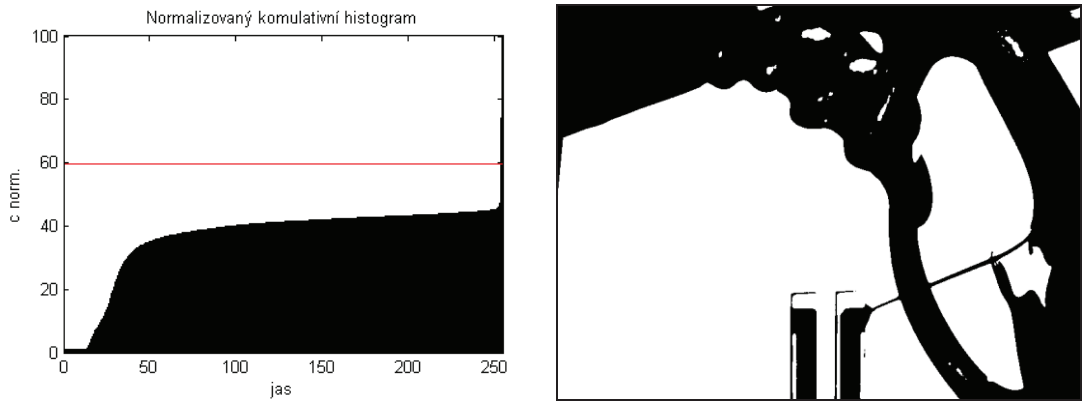
První krok, nalezení hladiny a vytvoření její masky, sestává z několika operací. Nejprve je provedeno prahování obrazu. Hledaný objekt je bílý – hladina leptacího roztoku. Při tomto zpracování je použito procentuální prahování, kdy je výchozím předpokladem procentuální zastoupení plochy objektů a pozadí ve scéně. Vzhledem k vlastnostem scény by však bylo možno použít i globální prahování nebo určení prahu na základě údolí v histogramu. Pro určení prahu při procentním prahování je využito kumulativního histogramu obrazu. Kumulativní histogram je integrálem, popřípadě sumou histogramu obrazu.

Histogram digitálního obrazu jako diskrétní funkce [2] a odpovídající kumulativní histogram:

$$h(r_k) = n_k \tag{22}$$

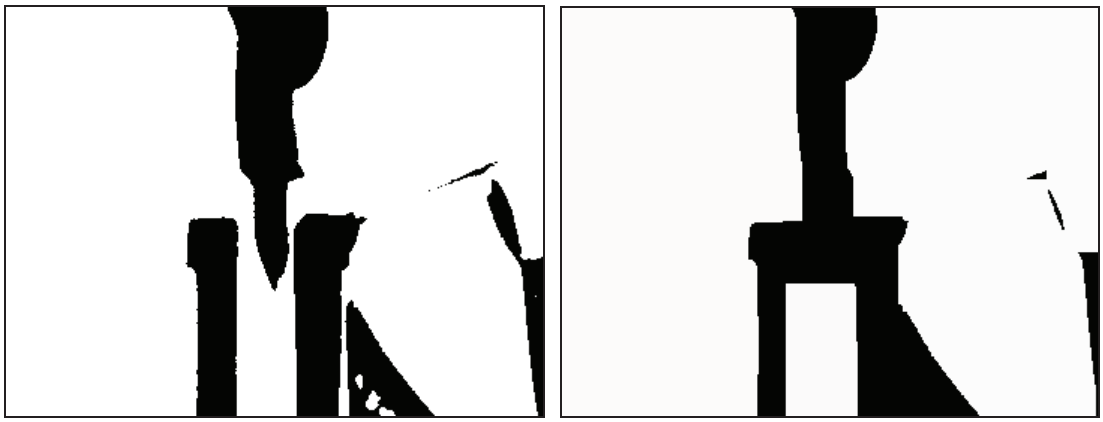
$$c(r_k) = \sum_{i=0}^{k} h(r_i) \tag{23}$$

Kde r_k je úroveň jasu, n_k je počet bodů v obraze s touto hodnotou jasu.



Obr. 25: Normalizovaný kumulativní histogram snímané scény a výsledek prahování pro 60% plochy objektů v obrazu

Určení prahu odpovídajícího procentnímu zastoupení objektů v obrazu lze provést přímo v procentech, pokud je provedena normalizace kumulativního histogramu v rozsahu hodnot 0-100. (Obr. 25)



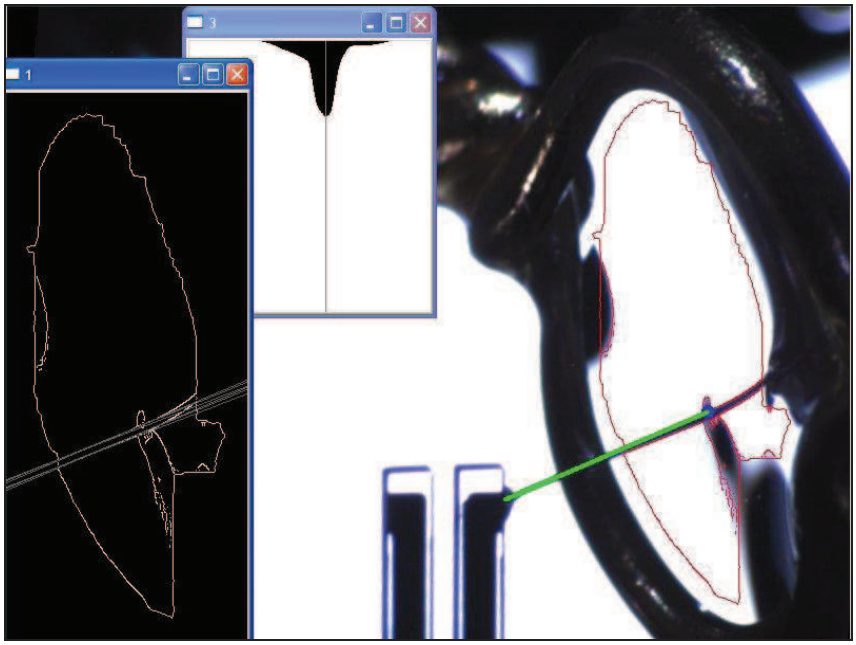
Obr. 26: Narušení prstence nosníkem a jeho oprava morfologickým uzavřením

Nosník může narušit leptací prstenec, pokud jej v obrazu protíná. Porušení leptacího prstence a spojení objektu hladiny a dalších bílých ploch je zabráněno aplikací morfologického uzavření na prahovaný obraz (viz Obr. 26). Opět je použito barvení k rozlišení jednotlivých objektů. Objekt hladiny je nalezen na základě velikosti a tvaru s pomocí popisu diskutovaného v kapitole 4.2.5.

V druhém kroku je nalezen drátek na základě obrazu hran a Houghovy transformace. Hladina neobsahuje rušivé objekty a pro nalezení správných hranic drátku postačuje předpokládaná orientace. Drátek by měl, vzhledem ke konstrukci stroje, svírat s osou nosníku úhel přibližně 22°.

Ve třetím kroku je nalezena oblast poklesu jasu v místě průniku drátku hladinou leptacího roztoku. Vstupními daty jsou hodnoty průměrného jasu v okolí drátku v oblasti hladiny. Minimum jasu v okolí drátku značí místo průniku.

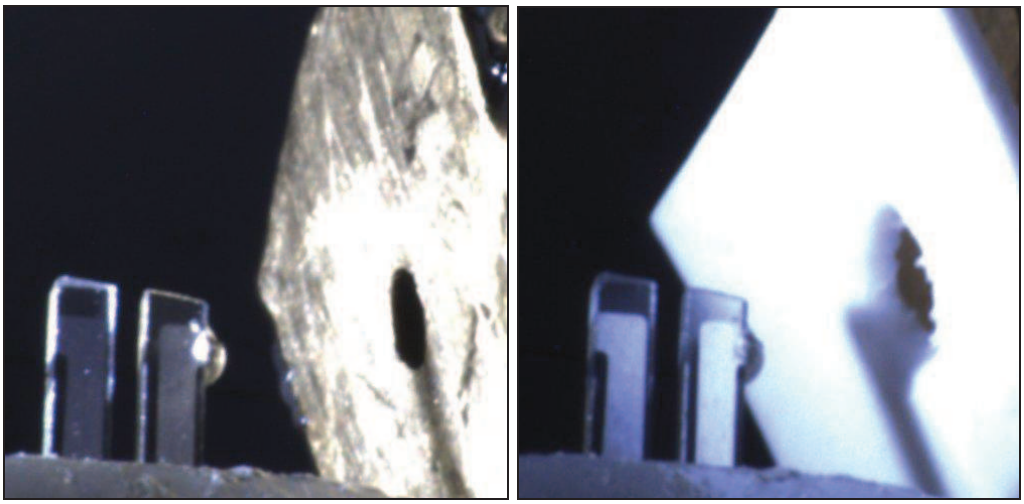
Čtvrtý krok je již pouhým přepočtem vzdálenosti v obrazu na jednotky skutečné vzdálenosti. Celý postup finálního zpracování ilustrují snímky na Obr. 27.



Obr. 27: Leptání hrotu – hrany získané Cannyho detektorem v oblasti hladiny a přímky získané Houghovou transformací (vlevo), pokles průměrného jasu v místě průchodu hladinou (nahoře) a výsledná vzdálenost od hladiny vyznačená v obrazu (vpravo)

4.9 ZMĚNA TECHNOLOGIE PŘI LEPTÁNÍ

Krátce před dokončením této práce došlo ke změně výrobní technologie posledního výrobního kroku. Hlavní změnou je zmenšení plochy hladiny leptacího roztoku, jak je patrné z Obr. 28. V důsledku této úpravy již není viditelný průhyb hladiny, a proto jej nelze použít pro lokalizaci místa průchodu drátku.



Obr. 28: Změna technologie a stín vržený nosníkem

Měření vzdálenosti zde předpokládá navedení drátku do středu leptacího otvoru. Vzdálenost k hladině je pak dána vzdáleností od nosníku ke středu leptacího otvoru. Navedení drátku předpokládá měření vzdáleností ve třech rozměrech, což úlohu komplikuje. Třetí rozměr není možno měřit pomocí zrcadla, jak tomu bylo při lepení hrotu. Důvodem nemožnosti použití zrcadla je nedostatek místa a malá hloubka ostrosti. Jedním z možných řešení, které se nabízí, je měření hloubky podle stínu vrženého nosníkem při nasvícení od hrany nosníku. Principiální uspořádání ilustruje Obr. 28 vpravo.

Vzhledem k nedávné změně výrobní technologie není konkrétní popis řešení, jeho implementace a výstupu součástí této práce.

5. ZÁVĚR

Práce pojednává o zpracování obrazu v jednotlivých fázích výroby hrotu pro mikroskop atomárních sil. Data získaná zpracováním obrazu slouží jako informace o vzájemné vzdálenosti jednotlivých montážních dílů, nebo technologických prvků.

Při řešení úlohy byly využity různé technické prostředky. Z pohledu počítačového vidění se jedná hlavně o snímací zařízení, jímž je kamera s vysokým rozlišením a makro objektivem, a programové prostředky, zejména knihovna Intel OpenCV zaměřená na zpracování obrazu.

Software vyvinutý v průběhu práce byl již od počátku koncipován takovým způsobem, aby byla zajištěna jeho přehlednost, jednoduchá modifikovatelnost a aby jeho struktura dodržovala logické a fyzické uspořádání výrobní linky a jejich částí. Pro dosažení této koncepce je softwarová implementace objektově orientovaná a provedena v jazyce C++. Jednotlivé objekty představují hardwarové části stroje, nebo softwarové bloky řešící specifické úlohy. Objekty jsou hierarchicky členěny do vrstev, kdy objekty vyšších vrstev využívají služeb nižších vrstev. Umístění nastavení aplikace a algoritmů zpracování do konfiguračních souborů dovoluje do určité úrovně modifikovat funkčnost, aniž by byl nutný zásah do kódu aplikace. Vedení záznamů o výrobě a funkčnosti stroje zjednodušuje odlaďování aplikace.

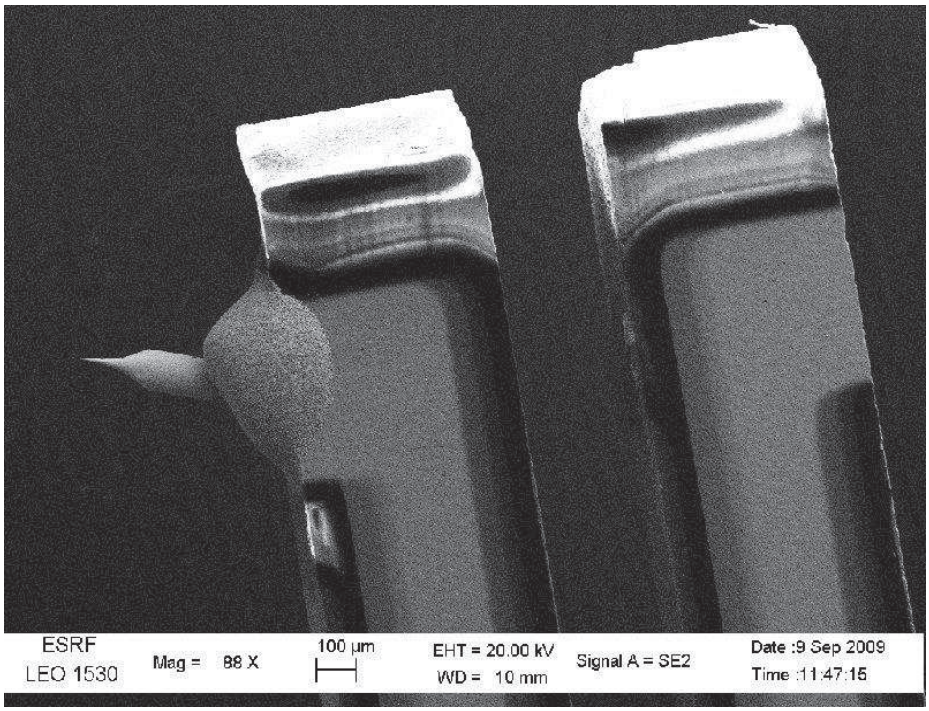
Při zpracování obrazu je hlavním krokem kalibrace kamery pomocí kalibračního vzoru. Díky kalibraci dochází ke korekci radiálního a tangenciálního zkreslení a korekci nesouběžnosti roviny scény s rovinou snímání. Z kalibrovaného obrazu lze zjistit velikost jednoho bodu obrazu v mikrometrech. Znalost těchto rozměrů je výchozím předpokladem pro přesné měření vzdáleností v obrazu. Dosažená rozlišitelnost je 10µm na jeden bod obrazu, nicméně tato hodnota je závislá na nastavení objektivu a rozlišení kamery.

V prvních dvou krocích výroby je nutno měřit vzdálenost mezi jednotlivými prvky ve třech osách. Pro toto snímání bylo využito zrcadla, protože jedno snímací zařízení neposkytuje všechny potřebné informace pro provedení měření. Zpracování obrazu je v tomto kroku založeno na adaptivním prahování pro segmentaci objektů jehly nebo drátku, rozlišení objektů barvením, popisu pomocí statistických momentů

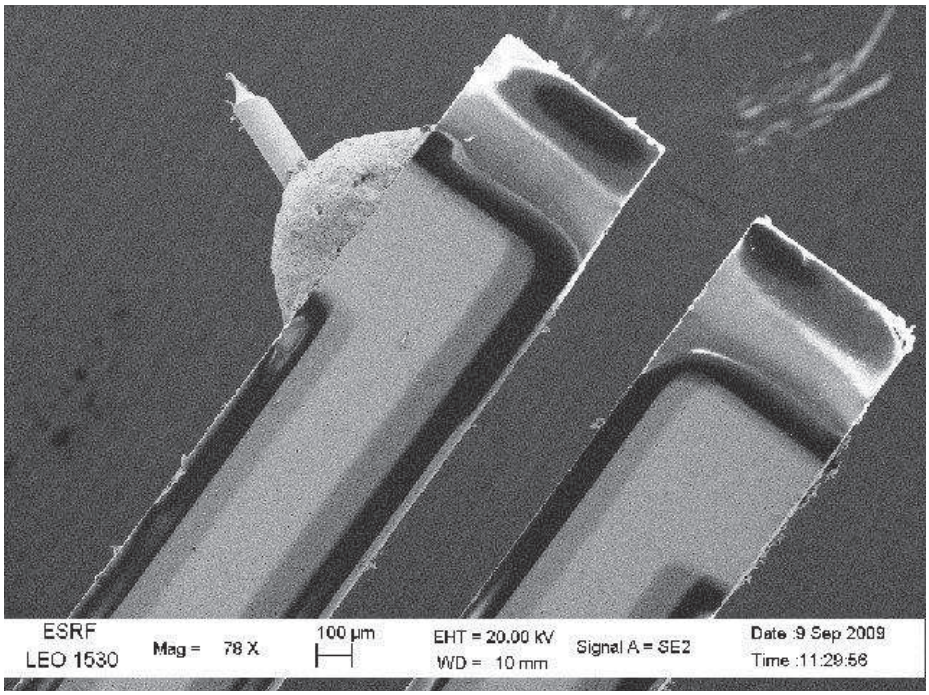
a radiometrických deskriptorů, Cannyho hranovém detektoru a Houghově transformaci. Algoritmy byly navrženy s ohledem na možnou změnu podmínek, hlavně osvětlení a polohy objektů, a v tomto smyslu také otestovány. Vykázaná kombinovaná úspěšnost dosahuje 85% pro jednotné nastavení algoritmů za proměnného osvětlení včetně vzniku nežádoucích stínů, přítomnosti cizích objektů a různé polohy jednotlivých montážních dílů.

Při třetím kroku, jímž je leptání hrotu, je měřena vzdálenost mezi nosníkem a hladinou leptacího roztoku. Bylo využito poklesu jasu v okolí průniku drátku do kapaliny. Algoritmus zpracování je založen na nalezení hladiny a místa poklesu průměrného jasu v okolí drátku. Nedávná změna výrobní technologie však nutně vyústí ve změnu zpracování ve třetím kroku. Tato nutná úprava není, z důvodu nedávné změny, zahrnuta v této práci.

Zařízení bylo schopno na základě dat získaných zpracováním obrazu vyrobit několik testovacích hrotů. Porovnání mezi výstupem stroje a člověka je na Obr. 29 a Obr. 30. Výstup stroje potvrzuje principiální správnost konceptu zpracování i výrobní linky. Vzhledem ke změně výrobní technologie a některým nutným konstrukčním změnám na straně hardwaru linky poskytnutého zadavatelem není v době psaní této práce stroj zcela dokončen a vývoj stále pokračuje.



Obr. 29: Hrot vyrobený strojem



Obr. 30: Hrot vyrobený ručně

6. LITERATURA

[1] BRADSKY, G., KAEHLER, A.: Learning OpenCV. O'reilly media, 2008. ISBN 978-0-596-51613-0.

[2] GONZALES, C.,R., WOODS, E.,R.: Digital Image Processing – second edition. Prentice Hall, 2001. ISBN 0-201-18075-8.

[3] Haußecker H., Geißler P.: Handbook of Computer Vision and Applications. San Diego: Academic press, 1999. ISBN 0-12-379770-5

[4] HLAVÁČ, V., ŠONKA, M.: Počítačové vidění. Praha: Grada, 1992 , ISBN 80-85424-67-3

[5] Hynčica, T.: *Příprava scény pro detekci elektronických součástek.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 57 s. Vedoucí diplomové práce Ing. Ilona Kalová, Ph.D.

[6] Intel.: OpenCV Reference Manual: 2001. [online]. Dostupné z www: <http://www.sciweavers.org/files/docs/307/opencv_manual.pdf> <http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf>

[7] OpenCV Wiki – online documentation and support for the OpenCV community. [online]. Dostupné z www: <http://opencv.willowgarage.com/wiki/Welcome>

[8] RUSS, C.,J.: The Image Processing Handbook – fifth edition. CRC Press, 2007. ISBN 0-8493-7254-2.

[9] Wikipedia The Free Encyclopedia: Atomic force microscope [online] Dostupné z www: < http://en.wikipedia.org/wiki/Atomic_force_microscope >

[10] Wikipedia The Free Encyclopedia: Computer Vision Wiki. [online]. Dostupné z www: <http://inperc.com/wiki/index.php?title=Main_Page>

Příloha 1

IMAGE PROCESSING IN SELFACTING PRODUCTION OF TIP FOR AFM MICROSKOPES (Image processing only)

Version 1.0
5/15/2010 6:13:00 PM

Table of Contents

Todo List

Class CAlgClass

Bug in units (not in um , but in 100ts of um). Not square chessboard (in calibration function). Water level processing is just test version, need to be changed according to technology. Some "small" memory leak exist during GetTipDistancesInUnits().

Member CAlgClass::GetWaterLevelDist (IplImage *inimage, float &x)

Only test version, need to be changed according to new technology.

Class CCamCalib

Make transformation faster.

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Class Documentation

_TAlgCfg Struct Reference

Configuration data.

`#include <CAlgClass.h>`

Public Attributes

- float **MountAngle**
- float **GlobalMaskUM**
- float **GlobalThresh**
- int **SmoothSize**
- int **CannyDilateS**
- bool **SubCanny**
- int **WireConvS**
- int **GlueConvS**
- int **CannyT1**
- int **CannyT2**
- int **CannyS**
- float **FModel** [4]
- int **FHTThresh**
- int **FHTCount**
- float **ForkAngle**
- float **ForkAngleTol**
- float **ForkAngleTopTol**
- int **MatchType**
- float **TopTestAreaTestUM**
- CvRect **ForkArea**
- int **WAproxOrder**
- float **WApStartP**
- float **WApStopP**
- float **WAngleTol**
- float **WMaxRatio**
- CvRect **WireArea**
- int **TAlgToUse**
- float **TopAngle**
- float **TAngleTol**
- int **THTThresh**
- int **THTCount**
- float **TParallelLineTol**
- int **RectS**
- int **LineS**

Detailed Description

Configuration data.

Member Data Documentation

int _TAlgCfg::CannyDilateS

Size of dilatation of canny image.

int _TAlgCfg::CannyS

Canny detector size.

int _TAlgCfg::CannyT1

Canny detector threshold 1.

int _TAlgCfg::CannyT2

Canny detector threshold 2.

int _TAlgCfg::FHTCount

HT lines count for selection.

int _TAlgCfg::FHTThresh

HT parameters space threshold.

float _TAlgCfg::FModel[4]

Fork borders spacing model in um.

float _TAlgCfg::ForkAngle

Fork angle in the image (range 0°-30° is expected, not tested for not vertical orientations).

float _TAlgCfg::ForkAngleTol

Fork angle parameter tolerance.

float _TAlgCfg::ForkAngleTopTol

Top angle tolerance.

CvRect _TAlgCfg::ForkArea

Area for looking for fork.

float _TAlgCfg::GlobalMaskUM

Size of the global adaptive threshold mask in um.

float _TAlgCfg::GlobalThresh

Threshold for adaptive thresholding.

int _TAlgCfg::GlueConvS

Size of convolution mask/thresholding area for glue.

int _TAlgCfg::LineS

Line thickness.

int _TAlgCfg::MatchType

Type of the correlation (0-5).

float _TAlgCfg::MountAngle

Angle of the holder.

int _TAlgCfg::RectS

Rectangle sizes.

int _TAlgCfg::SmoothSize

Size of smoothing mask.

bool _TAlgCfg::SubCanny

Substraction between thresholded image and canny.

int _TAlgCfg::TAlgToUse

Algorithm to use for top processing.

float _TAlgCfg::TAngleTol

Top angle tolerance.

int _TAlgCfg::THTCount

HT lines count for selection.

int _TAlgCfg::THTThresh

HT parameters space threshold.

float _TAlgCfg::TopAngle

Angle between top and horizontal direction.

float _TAlgCfg::TopTestAreaTestUM

Size of are to test top line is right (1um recommended).

float _TAlgCfg::TParallelLineTol

Parallel lines angle tolerance.

float _TAlgCfg::WAngleTol

Wire Angle tolerance to compare with TopAngle.

int _TAlgCfg::WAproxOrder

Order of line approximation.

float _TAlgCfg::WApStartP

Start of approximation in % of image width.

float _TAlgCfg::WApStopP

Stop of approximation in % of image width.

CvRect _TAlgCfg::WireArea

Area for looking for wires.

int _TAlgCfg::WireConvS

Size of convolution mask/thresholding area for wire.

float _TAlgCfg::WMaxRatio

Max ratio of wire bounding rectangle.

The documentation for this struct was generated from the following file:

- D:/!TIPS Project/Doxy test/CAlgClass.h

_TObjDesc Struct Reference

Object description.
`#include <CAlgClass.h>`

Public Attributes

- int **Area**
- int **Perimeter**
- float **Ratio**
- float **Orient**
- CvPoint **Possition**
- float **PixVarX**
- float **PixVarY**
- float **MeanPixX**
- float **MeanPixY**

Detailed Description

Object description.

Member Data Documentation

int _TObjDesc::Area

Size of the object.

float _TObjDesc::MeanPixX

Mean pixel count in X axis.

float _TObjDesc::MeanPixY

Mean pixel count in Y axis.

float _TObjDesc::Orient

Orientation of the object.

int _TObjDesc::Perimeter

Perimeter of the object.

float _TObjDesc::PixVarX

Variance in X axis.

float _TObjDesc::PixVarY

Variance in Y axis.

CvPoint _TObjDesc::Possition

Obj. rectangle points A B C D.

float _TObjDesc::Ratio

Width vs height of the object.

The documentation for this struct was generated from the following file:

- D:/!TIPS Project/Doxy test/CAlgClass.h

_TObjectA Struct Reference

Struct for acid processing.
`#include <CAlgClass.h>`

Public Attributes

- CvConnectedComp **comp**
  Basic parameters of object (size of bounding rectangle and surface).
- IplImage * **im**
  Corresponding object.

Detailed Description

Struct for acid processing.

The documentation for this struct was generated from the following file:

- D:/!TIPS Project/Doxy test/CAlgClass.h

_TValidPoints Struct Reference

Struct for image processing result.
`#include <CAlgClass.h>`

Public Attributes

- CvPoint **Wire1**
- CvPoint **Wire2**
- CvPoint **TipP1**
- CvPoint **TipP2**
- CvPoint **TipTop**
- CvPoint **WaterLevel**
- CvRect **ForkRealArea**
- bool **AllValid**

Detailed Description

Struct for image processing result.

Member Data Documentation

bool _TValidPoints::AllValid

Not used in code.

CvRect _TValidPoints::ForkRealArea

Rectangle Around Fork.

CvPoint _TValidPoints::TipP1

Fork point 1.

CvPoint _TValidPoints::TipP2

Fork point 2.

CvPoint _TValidPoints::TipTop

End point for top of the fork.

CvPoint _TValidPoints::WaterLevel

Point of water level and wire crossing.

CvPoint _TValidPoints::Wire1

End point of first wire.

CvPoint _TValidPoints::Wire2

End point of second wire.

The documentation for this struct was generated from the following file:

- D:/!TIPS Project/Doxy test/CAlgClass.h

CAlgClass Class Reference

Image processing class.
`#include <CAlgClass.h>`

Public Member Functions

- **CAlgClass** ()
  Constructor.
- virtual **~CAlgClass** ()
  Destructor.
- void **SetDebugMode** (bool mode)
  Debug mode flag set function.
- bool **GetDebugMode** ()
  Debug mode flag get function.
- IPErrors **GetTipDistances** (IplImage *inimage, int type, int &x, int &y, int &z)
  Distances for all axes, without units and transformation of image.
- IPErrors **DoThresholding** (IplImage *in, IplImage **out, int percentage)
  Percentage thresholding of the image.
- bool **SetTransformMatrix** ()
  Compensate scene rotation against motor axes.
- IPErrors **GetBWImage** (IplImage *in, IplImage **out)
  BW image conversion.
- IPErrors **ColorObjects** (IplImage *in, IplImage **out, CvConnectedComp **CompData, int &ColorCount, bool useMinSize=true, bool inverted=false)
  Objects coloring (marking).
- IPErrors **GetWaterLevelDist** (IplImage *inimage, float &x)
  Water level processing.
- IplImage * **MakeGraphFromVect** (IplImage *vector)
  Graph image from data vector (encoded in row IplImage).
- IplImage * **MakeSmoothGraph** (IplImage *vector, int size, bool smoothData=false)
  Smooth graph image from data vector (encoded in row IplImage).
- float **GetTopAngle** ()
  Access function for top angle.
- int **GetChCols** ()
  Access function for count of columns of chessboard.
- int **GetChRows** ()
  Access function for count of rows of chessboard.
- void **GetImSize** (int *width, int *heigth)
  Access function for set image size.
- int **GetImWidth** ()
  Access function for image width.
- int **GetImHeight** ()
  Access function for image height.
- string **GetPatternFile** ()
  Access function for pattern file name.

- void **SetGridUnits** (float units)
  Sets size of chessboard.
- IPErrors **Calibration** (IplImage *in, IplImage *patern)
  Calibration of camera based on chessboard pattern.
- IPErrors **GetConversion** (float &xconstant, float &yconstant)
  Conversion constants pixel-length.
- void **SetDimOfChessboard** (int rows, int cols)
  Sets number of internal chessboard points in both axes.
- IPErrors **ConvertToLenght** (int pixcount, float &lenght)
  Conversion pixels->length.
- IPErrors **GetTipDistancesInUnits** (IplImage *inimage, int type, float &x, float &y, float &z)
  Distances for all axes from transformed image (transformation is part of processing).
- IPErrors **TransformImage** (IplImage *inimage)
  Transformations of image.
- bool **PreprocSet** (IplImage *inimage, int type)
  Preprocessing settings function.
- bool **NoiseSet** (IplImage *inimage, int type)
  Noise reduction settings.
- bool **ThreshSet** (IplImage *inimage, int type)
  Threshold settings.
- bool **CannySet** (IplImage *inimage, int type)
  Canny settings.
- bool **FROISet** (IplImage *inimage)
  Fork ROI settings function.
- bool **WROISet** (IplImage *inimage)
  Wire ROI settings function.
- bool **TopSet** (IplImage *inimage)
  Top angle and tolerance set.

Detailed Description

Image processing class.

Class for image processing algorithms of the tip mounting, does not include image acquisition, processing only.

Author:

Miroslav Juhas

Date:

11.10.2009

Todo:

Bug in units (not in um , but in 100ts of um). Not square chessboard (in calibration function). Water level processing is just test version, need to be changed according to technology. Some "small" memory leak exist during **GetTipDistancesInUnits()**.

Constructor & Destructor Documentation

CAlgClass::CAlgClass ()

Constructor.

Begin operations, parameters for processing loaded from ini file, some other variables are set or computed. Calibration settings and log settings,...

See also:
INININAME

CAlgClass::~CAlgClass () `[virtual]`

Destructor.

Destructor. Saving all parameters used during processing - in case of change. In case of lost Ini file is this created with default values (loaded in constructor). Closing logs and ini files.

Member Function Documentation

IPErrors CAlgClass::Calibration (IplImage * *in*, IplImage * *pattern*)

Calibration of camera based on chessboard pattern.

Calibration.

Parameters:
**in* X input image for calibration.
**pattern* Y input pattern image.

Returns:
Some member of IPErrors struct, PROC_OK or first error which occurred.

bool CAlgClass::CannySet (IplImage * *inimage*, int *type*)

Canny settings.

Real time online Canny settings for the user.

Parameters:
**inimage* Image for preprocessing.
type Type of processing (WDIST or GDIST).

Returns:
False if inimage is NULL.

IPErrors CAlgClass::ColorObjects (IplImage * *in*, IplImage ** *out*, CvConnectedComp ** *CompData*, int & *ColorCount*, bool *useMinSize* = `true`, bool *inverted* = `false`)

Objects coloring (marking).

Function for marking (coloring) the objects in binary image. Input is expected like object 255, background 0. Maximum number of objects is limited by depth of the image. Function will accept depth of image up to 16bits, else return BAD_BITDEPTH. If number of objects exceed maximum count of objects will return MAX_COLOR_ERROR. Objects smaller then MinSize are filtered out when useMinSize parameter is true. Color 0 belongs to background.

See also:

IPErrors
MinSize

Parameters:

**in* Input image.
***out* Output image.
***CompData* Output data, CvConnectedComp, CvSeq pointer is not valid.
&ColorCount Count of objects found during processing.
useMinSize Flag for use minimal object size filtration.
inverted Are objects black (inverted - 0) or white (255)?

IPErrors CAlgClass::ConvertToLenght (int *pixcount*, float & *lenght*)

Conversion pixels->length.

Pixel count to length.

Parameters:

pixcount Pixel count.
&lenght Length corresponding to pixel count.

Returns:

Some member of IPErrors struct, PROC_OK or first error which occurred.

IPErrors CAlgClass::DoThresholding (IplImage * *in*, IplImage ** *out*, int *percentage*)

Percentage thresholding of the image.

Percentage thresholding for the image based on histogram. The threshold is set with corespondence to percentage of points in image belogs under the brightness threshold. Used e.g when object is dark or bright and have known size in the image.

Parameters:

**in* Input image.
***out* Output binary image.
percentage Size of the objects in percentago of the image (or inverse).

bool CAlgClass::FROISet (IplImage * *inimage*)

Fork ROI settings function.

Real time online fork ROI settings for the user.

Parameters:

**inimage* Image for preprocessing.

Returns:

False if inimage is NULL.

IPErrors CAlgClass::GetBWImage (IplImage * *in*, IplImage ** *out*)

BW image conversion.

BW conversion function. Out must be passed NULL (prevent memory leak when passed non empty image).

Parameters:

**in* Input IplImage image pointer.
***out* Output BW image.

IPErrors CAlgClass::GetConversion (float & *xconstant*, float & *yconstant*)

Conversion constants pixel-length.

Length of one pixel in both axis.

Parameters:

&xconstant X constant.
&yconstant Y constant.

Returns:

Some member of IPErrors struct, PROC_OK or first error which occurred.

IPErrors CAlgClass::GetTipDistances (IplImage * *inimage*, int *type*, int & *x*, int & *y*, int & *z*)

Distances for all axes, without units and transformation of image.

Give a distances in pixels in all motor axes. Image calibration is not used!. Direct use not recommended. Is aimed to use in combination with image calibration and conversion functions.

See also:

GetTipDistancesInUnits(IplImage *inimage,float &x,float &y,float &z)
CCamCalib

Parameters:

**inimage* Input image.
type Type of processing (glue or wire).
&x Returned X distance.
&y Returned Y distance.
&z Returned Z distance.

IPErrors CAlgClass::GetTipDistancesInUnits (IplImage * *inimage*, int *type*, float & *fx*, float & *fy*, float & *fz*)

Distances for all axes from transformed image (transformation is part of processing).

Same like GetTipDistances but returned distances are in units.

See also:

GetTipDistances

float CAlgClass::GetTopAngle () `[inline]`

Access function for top angle.

Warning:

Top angle for IP and for montage is not the same.

IPErrors CAlgClass::GetWaterLevelDist (IplImage * *inimage*, float & *xdistance*)

Water level processing.

Water level processing function. Fork point is taken from last wire processing (no changes in fork position are expected). Water level inside the ring is taken like a mask (after some preprocessing) for image processing. We are looking for change in brightness around wire when crossing water level. Returned distances are in um.

Parameters:

**inimage* Input IplImage.
&xdistance Returned distance between fork and water level in um.

Todo:

Only test version, need to be changed according to new technology.

Warning:

Test version only, old implementation - technology changed.

IplImage * CAlgClass::MakeGraphFromVect (IplImage * *vector*)

Graph image from data vector (encoded in row IplImage).

Image of graph from vector of values. Help function for debug.

Parameters:

**vector* Vector of values encoded in row if the image. Byte values (8bit image).

Returns:

Image of the graph corresponding to values

IplImage * CAlgClass::MakeSmoothGraph (IplImage * *vector*, int *size*, bool *smoothData* = `false`)

Smooth graph image from data vector (encoded in row IplImage).

Image of smooth graph from vector of values. Help function for debug.

Parameters:

**vector* Vector of values encoded in row if the image. Byte values (8bit image).
size Size of smoothing, must be odd.
smoothData Change the imput too?

Returns:

Image of the smooth graph corresponding to values.

bool CAlgClass::NoiseSet (IplImage * *inimage*, int *type*)

Noise reduction settings.

Real time online noise reduction settings for the user.

Parameters:

**inimage* Image for preprocessing.
type Type of processing (WDIST or GDIST).

Returns:

False if inimage is NULL.

bool CAlgClass::PreprocSet (IplImage * *inimage*, int *type*)

Preprocessing settings function.

Real time online preprocessing settings for the user.

Parameters:

**inimage* Image for preprocessing.
type Type of processing (WDIST or GDIST).

Returns:

False if inimage is NULL.

void CAlgClass::SetGridUnits (float *units*)

Sets size of chessboard.

Calibration. Set distances between line grids.

Parameters:

units Size of the square of chessboard in units.

bool CAlgClass::ThreshSet (IplImage * *inimage*, int *type*)

Threshold settings.

Real time online threshold settings for the user.

Parameters:

**inimage* Image for preprocessing.
type Type of processing (WDIST or GDIST).

Returns:

False if inimage is NULL.

bool CAlgClass::TopSet (IplImage * *inimage*)

Top angle and tolerance set.

Real time online top angle settings for the user.

Parameters:

**inimage* Image for preprocessing.

Returns:

False if inimage is NULL.

IPErrors CAlgClass::TransformImage (IplImage * *inimage*)

Transformations of image.

Image transformation according to calibration.

Parameters:

**inimage* Image for transformation.

Returns:

Some member of IPErrors struct, PROC_OK or first error which occurred.

bool CAlgClass::WROISet (IplImage * *inimage*)

Wire ROI settings function.

Real time online wire ROI settings for the user.

Parameters:

**inimage* Image for preprocessing.

Returns:

False if inimage is NULL.

The documentation for this class was generated from the following files:

- D:/!TIPS Project/Doxy test/CAlgClass.h
- D:/!TIPS Project/Doxy test/CAlgClass.cpp

CCamCalib Class Reference

Camera calibration class.
`#include <CCamCalib.h>`

Public Member Functions

- **CCamCalib** ()
- **~CCamCalib** ()
- void **SetGridUnits** (float units)
  Size of one square of chessboard in units.
- void **SetDimOfChessboard** (int rows, int cols)
  Count of internal chessboard points in both axes.
- bool **Calibration** (IplImage *in, IplImage *pattern, bool showres)
  Calibration itself.
- bool **GetConversion** (float &xconstatn, float &yconstant)
  Get conversion constants.
- bool **ConvertToLenght** (int pixcount, float &lenght)
  Pixel count to length.
- bool **TransformImage** (IplImage *in)
  Image transformation.

Detailed Description

Camera calibration class.

Class for conversion between pixel and length. Calibration from a picture of grid of known size. (CHESSBOARD)

Author:
Miroslav Juhas

Date:
30.4.2010

Note:
For use with image, no with camera,

Warning:
Resolution of camera in both axes and resolution of the calibration pattern must be same(different resolution not implemented) Missing detection of bad pattern or segmentation.

Todo:
Make transformation faster.

Constructor & Destructor Documentation

CCamCalib::CCamCalib ()

Constructor.

CCamCalib::~CCamCalib ()

Destructor.

Member Function Documentation

bool CCamCalib::Calibration (IplImage * *image*, IplImage * *pattern*, bool *showres* = `false`)

Calibration itself.

Chessboard and OpenCV based camera calibration.

Parameters:

**image* Image of chessboard for calibration (taken by camera).
**pattern* Pattern image.
showres Show results for debugging.

Returns:

False if failed to find interest points or failed to calibrate at all.

bool CCamCalib::ConvertToLenght (int *pixcount*, float & *lenght*)

Pixel count to length.

Convert distance in pixels to units.

Parameters:

pixcount Pixel count.
&lenght Corresponding length.

Returns:

ColOk variable. Depend on calibration result.

bool CCamCalib::GetConversion (float & *xconstant*, float & *yconstant*)

Get conversion constants.

Size of one pixel in units. For X and Y image dimensions, but the different resolution is not implemented.

Parameters:

&xconstant Size in X.
&yconstant Size in Y.

Returns:

ColOk variable. Depend on calibration result.

void CCamCalib::SetGridUnits (float *units*)

Size of one square of chessboard in units.

Size of the calibration pattern in units.

Parameters:

units Size in units.

bool CCamCalib::TransformImage (IplImage * *in*)

Image transformation.

Image transformation according to calibration.

Parameters:

**in* Image for transformation.

Returns:

False if calibration isnt done.

The documentation for this class was generated from the following files:

- D:/!TIPS Project/Doxy test/CCamCalib.h
- D:/!TIPS Project/Doxy test/CCamCalib.c

Index

Please see electronic version.